Wydział Elektroniki

PRACA DOKTORSKA

Sterowanie wydajnością mikroprocesorów przy ograniczeniu temperaturowym

Bartosz Wojciechowski

Promotor: dr hab. inż. Janusz Biernat, prof. P.Wr.

krótkie streszczenie:
Praca obejmuje zagadnienia związane z modelowaniem oraz dynamicznym zarządzaniem temperaturą procesorów wielordzeniowych

Wrocław 2012

Spis treści

Wykaz oznaczeń i najczęściej używanych skrótowców

A	—	pole powierzchni
V	—	objętość
c	—	ciepło właściwe materiału
h	—	współczynnik przenikania ciepła
k	—	przewodność cieplna
ρ	—	gęstość materiału
t	—	czas
P	—	moc
T	—	temperatura
Q	—	ciepło
I	—	macierz jednostkowa
$e^{\mathbf{A}}$	—	eksponenta macierzy $\mathbf{A}$
N_c	—	liczba rdzeni w procesorze
f_i	—	częstotliwość taktowania i-tego rdzenia procesora
$f_{\max}$	—	maksymalna częstotliwość taktowania procesora
s_i	—	ustawienie szybkości danego rdzenia procesora, para (f_i,V_i)
$\mathbf{S}$	—	wektor szybkości poszczególnych rdzeni procesora
$\mathbf{\Phi} = \{f^1, .., f^n\}$	—	wektor dostępnych częstotliwości rdzeni procesora
R_{amb}	—	rezystancja termiczna pomiędzy radiatorem a otoczeniem
R_{tot}	—	całkowita rezystancja termiczna pomiędzy procesorem, a otoczeniem

N_{RC} — rozmiar modelu termicznego, liczba węzłów sieci

$\mathbf{T}$ — kolumnowy wektor temperatur, wektor stanu modelu termicznego

T_i — temperatura i-tego węzła modelu termicznego

$T_{\max}$ — maksymalna dozwolona temperatura pracy procesora

T_{th}^{mig} — temperatura progowa migracji

T_{amb} — temperatura otoczenia

T_{pred}^{i} — prognozowana temperatura i-tego bloku procesora

t_s — długość kroku algorytmu prognozy

ε_i — błąd prognozy temperatury i-tego bloku procesora

k_ε — współczynnik korekty modelu termicznego na podstawie sumarycznego błędu prognozy temperatury rdzeni

$\mathbf{A}$ — macierz przejścia modelu termicznego

$\mathbf{B}$ — macierz wejść modelu termicznego

$\mathbf{C}$ — macierz pojemności

$\mathbf{D}$ — macierz konduktancji

$\mathbf{E}$ — macierz przejść w rozwiązaniu równania stanu

$\mathbf{F}$ — macierz wejść w rozwiązaniu równania stanu

c_i — pojemność cieplna i-tego węzła w modelu termicznym

r_{ij} — rezystancja termiczna pomiędzy i-tym, a j-tym elementem modelu termicznego

I_{leak} — prąd upływu

P_{dyn}^{i},P_{stat}^{i} — moc dynamiczna i statyczna rozpraszana przez i-ty blok procesora

P_k — moc rozpraszana przez cały komputer

P_{idle} — moc rozpraszana przez procesor w stanie bezczynności

$\mathbf{P}$ — wektor kolumnowy mocy rozpraszanych przez elementy procesora

k_A,k_B,k_C,k_D,k_E — współczynniki regresji

N_{th} — liczba wątków uruchomionych na komputerze

CMOS — (ang. *Complementary metal-oxide-semiconductor*) – technologia wytwarzania układów cyfrowych z komplementarnych par tranzystorów planarnych

CMP — (ang. *Chip Multi Processing*) – zwielokrotnienie rdzeni w obrębie jednego procesora

CPU — (ang. *Central Processing Unit*) – procesor komputera

DVFS — (ang. *Dynamic Voltage and Frequency Scaling*) – dynamiczne skalowanie napięcia zasilania i częstotliwości pracy procesora, bądź jego elementu

HPC — (ang. *High Performance Computing*) – obliczenia wysokiej wydajności

ILP — (ang. *Instruction Level Parallelism*) – równoległość poziomu instrukcji. Współczynnik określający ile instrukcji w programie może być wykonywanych jednocześnie

IPC — (ang. *Instructions Per Cycle*) – współczynnik określający liczbę instrukcji przetwarzanych podczas jednego cyklu zegara procesora

MOSFET — (ang. *Metal Oxide Semiconductor Field Effect Transistor*) – tranzystor polowy z izolowaną bramką

MSR — (ang. *Model Specific Register*) – rejestr kontrolujący wybraną funkcję danego procesora, jak monitorowanie wydajności i temperatury lub zarządzanie mocą

PMC — (ang. *Performance Monitoring Counter*) – licznik zdarzeń na poziomie mikroarchitektury procesora służący do monitorowania wydajności procesora

SMT — (ang. *Simultaneous MultiThreading*) – współbieżna wielowątkowość; architektury SMT cechują się możliwością jednoczesnego wykonywania więcej niż jednego wątku

TIM — (ang. *Thermal Interface Material*) – materiał zwiększający konduktancję termiczną na styku dwóch warstw

TDP — (ang. *Thermal Design Point/Power*) – zakładany maksymalny poziom mocy rozpraszanej przez procesor, wyznacza wymaganą skuteczność układu chłodzenia

VLSI — (ang. *Very-Large-Scale Integration*) – budowa układów elektronicznych składających się z bardzo wielu tranzystorów w pojedynczym układzie scalonym

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

1.1 Motywacja

W ostatnich latach temperatura stała się jednym z najważniejszych czynników ograniczających wydajność procesorów [1]. Zmniejszające się wymiary nowych układów produkowanych w technologii CMOS powodują zwiększanie się gęstości mocy na powierzchni procesorów. Nierównomierne rozpraszanie mocy w procesorze skutkuje powstawaniem gradientów termicznych i w konsekwencji, gorących punktów. Wzrost temperatury prowadzi do obniżenia niezawodności układu. Natężenie zjawisk takich jak elektromigracja czy starzeniowe przebicie dielektryka jest wykładniczo zależne od temperatury. Stąd pojęcie mikroprocesorów ograniczonych przez temperaturę, w których czynnikiem limitującym wydajność jest temperatura złącza w tak zwanych gorących punktach (ang. *hot spots*).

Wraz ze wzrostem gęstości mocy na powierzchni mikroprocesorów zwiększają się koszty chłodzenia. Gęstość mocy na powierzchni współczesnych procesorów przekracza 100 W/cm^2. Powyżej limitu skuteczności chłodzenia powietrzem (około 150 W/cm^2 [1]) konieczne jest wykorzystanie innych, bardzo kosztownych metod chłodzenia, na przykład chłodzenia wodnego. Chłodzenie powietrzem, oprócz ograniczonej wydajności ma również wadę w postaci dużej głośności, co ogranicza walory użytkowe i stanowi kolejny argument za ograniczaniem temperatury procesorów.

Oprócz negatywnego wpływu na niezawodność, wysoka temperatura ma wpływ na zużycie energii. Zmniejszanie się wymiarów tranzystorów w nowych technologiach pociąga za sobą negatywne efekty w postaci wzrostu natężenia prądu upływu. Natężenie zjawiska wyciekania elektronów od źródła do drenu wyłączonych tranzystorów MOS

jest wykładniczo zależne od temperatury. W ten sposób powstaje dodatnie sprzężenie zwrotne pomiędzy temperaturą a mocą statyczną rozpraszaną przez procesor. Z uwagi na duży poziom wariancji procesowej i małe wymiary bramek ten problem nasilił się w technologiach poniżej 100 nm. Obciążenie procesorów podlega dodatkowo dużym zmianom w czasie. W zależności od warunków otoczenia, oraz aktualnie przetwarzanych zadań zmienia się ilość rozpraszanej energii, oraz temperatura w różnych punktach procesora.

Klasyczne podejście do projektowania układów cyfrowych przy zapewnieniu określonego poziomu niezawodności wymaga przyjęcia założenia o najgorszych warunkach pracy i maksymalnym obciążeniu. W praktyce dużą zmienność obciążenia procesora daje się wykorzystać dla polepszenia jego średniej wydajności bez jednoczesnego zwiększania kosztów. Dostosowywanie szybkości pracy procesora do aktualnego obciążenia i ograniczeń temperatury nazywa się dynamicznym zarządzaniem temperaturą (ang. *Dynamic Thermal Management – DTM*). Mechanizmy DTM, takie jak skalowanie napięcia zasilania i częstotliwości (DVFS) oraz migracja zadań pozwalają na redukcję gradientów termicznych w czasie i przestrzeni. Chroni to przed występowaniem nadmiernej temperatury w gorących punktach.

W procesorach wielordzeniowych nadmiarowe zasoby obliczeniowe znajdują się naturalnie w postaci kolejnych rdzeni procesora. Umożliwia to migrację zadania z przegrzewającej się jednostki wykonawczej do chłodniejszej. Regularna struktura ułatwia zarządzanie zasobami obliczeniowymi uwzględniające temperaturę na poziomie systemu operacyjnego. Nowe technologie pozwalają na niezależną kontrolę szybkości działania poszczególnych rdzeni procesorów. Aktualnym problemem jest efektywne zarządzanie zasobami obliczeniowymi procesorów wielordzeniowych z uwzględnieniem temperatury ich pracy, w szczególności pod ograniczeniem maksymalnej dopuszczalnej temperatury.

Efektywne strategie i algorytmy zarządzania temperaturą procesora wymagają modeli termicznych procesorów o złożoności pozwalającej na wykorzystanie ich w czasie bieżącego działania systemu przy minimalnym narzucie na wydajność komputera. Dokładna prognoza temperatury pozwala na podjęcie właściwej decyzji i wykorzystanie wolnych zasobów, co przekłada się na zwiększenie wydajności przy ograniczeniu maksymalnej temperatury.

Dynamiczne zarządzanie temperaturą na seryjnie produkowanych (standardowych) procesorach pozwoli ograniczyć koszty związane z ich wytwarzaniem oraz chłodze-

niem, a także zwiększyć niezawodność w zmiennych warunkach pracy bez strat wydajności.

Problemy z utrzymaniem temperatury na odpowiednim poziomie będą dodatkowo zaostrzone w wyniku integracji układów cyfrowych w trzecim wymiarze [2]. Za ważnością podejmowanego w niniejszej rozprawie tematu przemawia również duża liczba publikacji dotyczących temperatury procesorów [3–5]. W ostatniej dekadzie w materiałach najważniejszych konferencji dotyczących projektowania i testowania układów cyfrowych, architektury komputerów oraz obliczeń wysokiej wydajności pojawiło się wiele publikacji podejmujących temat dynamicznego zarządzania temperaturą. Wynika to z konieczności uwzględnienia zjawisk związanych z temperaturą w projektowaniu procesorów.

1.2 Cel i teza pracy

Celem pracy jest opracowanie metody zarządzania temperaturą procesorów, której wynikiem będzie zwiększenie wydajności przetwarzania wielu zadań na procesorze wielordzeniowym przy ograniczeniu maksymalnej temperatury.

Realizując to zamierzenie zaproponowano algorytm sterowania działaniem procesora oparty o prognozę temperatury w powiązaniu z informacją o aktywności poszczególnych rdzeni, skalowanie napięcia i częstotliwości oraz migrację wątków. Niezbędną podstawą tego algorytmu było opracowanie modelu temperatury, rozpraszanej mocy oraz wydajności procesora wielordzeniowego, a także określenie wpływu mechanizmów dynamicznego zarządzania temperaturą na powyższe parametry. Jednym z efektów dodatkowych było utworzenie wysokopoziomowego modelu aktywności programów, przeznaczonego do szybkich symulacji termicznych procesorów wielordzeniowych.

Przeprowadzone badania i analizy doprowadziły do sformułowania następującej tezy niniejszej rozprawy:

Wykorzystanie informacji o przebiegu programów
do sterowania działaniem procesora wielordzeniowego
pozwala na zwiększenie wydajności przetwarzania
pomimo ograniczenia maksymalnej temperatury.

1.3 Struktura pracy

Przedmiotem pracy jest metoda sterowania pracą procesorów wielordzeniowych maksymalizująca wydajność pod ograniczeniem temperaturowym oraz efektywne modelowanie termiczne dynamicznego zachowania procesorów. Praca zawiera wyniki symulacji oraz eksperymentów przeprowadzonych na komputerze z czterordzeniowym procesorem Intel Core 2 Q9400.

W rozdziale drugim zawarto opis zagadnień związanych z temperaturą w procesorach. Przedstawiono mechanizmy wytwarzania i przekazywania energii cieplnej, wpływ temperatury na pracę procesorów oraz najnowsze prace na temat modelowania temperatury w procesorach, sterowania pracą procesora z uwzględnieniem temperatury oraz prognozowania i szacowania wydajności procesorów wielordzeniowych pod ograniczeniem maksymalnej temperatury. Rozdział trzeci zawiera opis zagadnień związanych z modelowaniem wydajności i temperatury procesora, wykonanych pomiarów oraz wykorzystywanego modelu termicznego. W rozdziale czwartym zaprezentowano nowy algorytm prognozy temperatury na podstawie aktywności procesora i jego modelu termicznego. Przedmiotem rozdziału piątego jest sterowanie wydajnością procesora przy ograniczeniu temperaturowym. Rozdział ten zawiera opis nowego heurystycznego algorytmu dynamicznego zarządzania temperaturą procesora oraz eksperymentalnej weryfikacji jego skuteczności.

Rozdział 2

Temperatura w procesorach

W niniejszym rozdziale omówione zostaną podstawowe prawa dotyczące przekazywaniem ciepła w procesorach, jego źródła oraz wpływ na niezawodność i działanie procesorów. Następnie opisane zostaną znane metody sterowania temperaturą na poziomie architektury procesorów, metody dynamicznego zarządzania temperaturą oraz prognozy temperatury w procesorach wielordzeniowych.

2.1 Temperatura w procesorach

Podstawowym źródłem ciepła we współczesnych procesorach jest ładowanie się i rozładowywanie pojemności podczas przełączania tranzystorów oraz występowanie krótkotrwałych prądów zwarcia w momencie, gdy sieci nMOS i pMOS są częściowo załączone [6]. Oba te czynniki są przyczyną rozpraszania mocy dynamicznej w urządzeniach wykonanych w technologii CMOS. Dodatkowym źródłem ciepła jest moc statyczna, związana głównie z prądem upływu oraz prądem podprogowym. Intensywność obu zjawisk zależy od szeregu czynników, wśród których jednym z najistotniejszych jest temperatura.

2.1.1 Ciepło

Ciepło jest procesem przekazywania energii wewnętrznej pomiędzy obiektami termodynamicznymi. Są znane trzy mechanizmy transferu ciepła: przewodzenie, konwekcja oraz radiacja. Przewodzenie ciepła w procesorach opisuje prawo Fouriera, według

którego gęstość przewodzonego strumienia ciepła jest wprost proporcjonalna do gradientu temperatury [7].

$$\frac{dQ}{dt} = -kA\frac{dT}{dx}, \tag{2.1}$$

Szybkość przewodzenia ciepła $\frac{dQ}{dt}$ w kierunku prostopadłym do warstwy o powierzchni A i grubości x zależy od różnicy temperatur po obu stronach tej warstwy. W równaniu (2.1) T to temperatura w kelwinach, a k jest współczynnikiem przenikalności cieplnej. Jednostką współczynnika przenikalności cieplnej jest [W/(m·K)]. Wartość współczynnika przenikalności cieplnej zależy od właściwości materiału przewodzącego. Minus przed współczynnikiem przenikalności cieplnej wynika z tego, że przepływ ciepła odbywa się przeciwnie do gradientu temperatury.

W komputerach chłodzonych powietrzem energia termiczna odprowadzana jest z obudowy procesora przez radiator, zwykle z wymuszonym przepływem powietrza. Konwekcja polega na przekazywaniu energii cieplnej do gazu lub cieczy poruszającej się obok chłodzonego ciała. Zjawisko konwekcji opisuje prawo chłodzenia Newtona, zgodnie z którym szybkość transferu ciepła na skutek konwekcji jest proporcjonalna do różnicy pomiędzy temperaturą chłodzonej powierzchni T_s a temperaturą chłodzącego gazu lub cieczy T_{amb}

$$\frac{dQ}{dt} = hA(T_s - T_{amb}). \tag{2.2}$$

Współczynnik przenikania ciepła h określa szybkość wymiany energii poprzez konwekcję. Jego jednostką jest [W/(m^2·K)], a wartości wyznacza się eksperymentalnie. Wartość współczynnika przenikania ciepła h zależna jest od przepływu chłodzącego gazu lub cieczy oraz geometrii chłodzonego obiektu. W obecności wymuszonej konwekcji radiacja ma minimalne znaczenie, a uwzględnienie jej możliwe jest poprzez wyznaczenie wspólnego współczynnika przenikania ciepła uwzględniającego to zjawisko [7].

Pojemność cieplna poszczególnych elementów procesora, obudowy i radiatora zależy od ich masy oraz ciepła właściwego materiałów z których są wykonane. Ciepło właściwe c substancji to ciepło potrzebne do zmiany temperatury jednostki masy substancji o jeden Kelvin

$$c = \frac{\Delta Q}{m \cdot \Delta T}. \tag{2.3}$$

Jednostką ciepła właściwego w układzie SI jest [J/(kg·K)]. Pojemność cieplna układu chłodzenia jest zazwyczaj o 3 rzędy większa niż pojemność cieplna samego krzemu.

Spowodowane jest to dużą masą radiatora (dochodzącą do 1 kilograma w wysokowydajnych układach chłodzenia powietrzem) oraz tym, że radiatory i obudowy tworzy się z metali o większym cieple właściwym niż krzem.

Moc statyczna, której podstawowym źródłem jest prąd upływu (ang. *leakage current*), jest wykładniczo zależna od temperatury. Głównymi źródłami upływu w zakresach temperatur spotykanych w procesorach jest tunelowanie przez izolator bramki oraz prąd podprogowy, czyli prąd płynący od źródła do drenu tranzystora polowego pracującego w zaporowej części charakterystyki. Prąd podprogowy I_{sub} jest silnie zależny od wymiarów kanału tranzystora oraz temperatury [8]:

$$I_{sub} = k_{tech} \left(\frac{W}{L}\right) 10^{-\frac{V_T}{S}}, \tag{2.4}$$

gdzie W i L to odpowiednio szerokość i długość kanału tranzystora, V_T to napięcie progowe tranzystora, a $S = 2.3nk_BT/q$ jest parametrem określającym spadek napięcia progowego skutkujący dziesięciokrotnym wzrostem prądu upływu. Napięcie progowe jest w przybliżeniu, liniowo zależne od temperatury [6]:

$$V_T(T) = V_T(T_0) - k_{vt}(T - T_0). \tag{2.5}$$

Dodatnie sprzężenie zwrotne pomiędzy temperaturą a mocą rozpraszaną powoduje ryzyko ucieczki termicznej. Bezpieczeństwo procesora musi być więc zapewnione zarówno podczas projektowania z uwzględnieniem zależności mocy statycznej od temperatury, jak i podczas pracy poprzez mechanizmy ograniczające możliwość pracy w wysokich temperaturach. Z tego powodu przy zmniejszaniu wymiarów tranzystorów konieczne było zwiększenie rezystancji izolatora bramki [9].

Temperatura wpływa na wydajność procesora również przez ograniczanie mobilności nośników prądu μ, wg wzoru [6]:

$$\mu(T) = \mu(T_0) \left(\frac{T}{T_0}\right)^{-k_\mu}, \tag{2.6}$$

gdzie $\mu(T)$ jest mobilnością nośników prądu w temperaturze T, a k_μ jest parametrem technologicznym przyjmującym zwykle wartości w zakresie 1,2–2. Oznacza to, że wraz ze zwiększaniem temperatury, na ścieżkach krytycznych mogą pojawić się opóźnienia. Stąd konieczne jest uwzględnienie dopuszczalnego zakresu temperatur w projektowaniu układu.

Konkretnym przykładem procesorów o wydajności ograniczonej przez temperaturę i moc są układy z technologiami `Turbo Boost` (`Intel`) oraz `Turbo Core` (`AMD`). Procesory konstruowane z tymi technologiami mogą działać z podwyższoną częstotliwością

pracy jednego bądź kilku rdzeni jeżeli bieżąca temperatura, szacowany pobór prądu i szacowana moc rozpraszana nie przekracza ustalonych wartości [10]. Nowsze procesory firmy `Intel` zwiększają wydajność do poziomu który skutkuje rozpraszaniem mocy powyżej maksymalnego poziomu jaki układ chłodzenia powinien być w stanie odprowadzić – TDP (ang. *Thermal Design Point*), jeśli chwilowo temperatura procesora i radiatora nie jest wysoka [11]. Wykorzystywana jest przy tym duża pojemność cieplna układu chłodzenia. Z oczywistych względów nie prowadzi to do znacznej poprawy wydajności przy długotrwałym przetwarzaniu zadań. Metoda ta jest oparta na szacowaniu temperatury przez procesor.

Wpływ temperatury i modelowanie niezawodności procesorów

Temperatura pracy wpływa w dużym stopniu na niezawodność działania procesorów. Jest to spowodowane szeregiem mechanizmów, których intensywność zależy od bieżącej temperatury [12]. Jednym z nich jest elektromigracja, która polega na przemieszczaniu się atomów metalizacji w warstwie ścieżek przewodzących pod wpływem przepływającego prądu elektrycznego i może prowadzić do otwarcia połączeń i zwarć. Średni czas do uszkodzenia spowodowanego przez elektromigrację może być opisany wzorem [12]:

$$MTTF \propto J^{-n_{EM}} * e^{\frac{E_{a_{EM}}}{kBT}}. \tag{2.7}$$

We wzorze (2.7) k jest stałą Boltzmanna, T temperaturą w skali Kelwina, $\mathrm{E}_{a_{EM}}$ oraz n_{EM} to stałe zależne od zastosowanych materiałów, a J oznacza natężenie prądu płynącego w ścieżce.

Starzeniowe przebicie dielektryka (ang. *Time Dependent Dielectric Breakdown – TDDB*) jest mechanizmem, który powoduje powstawanie przewodzących ścieżek w warstwie dielektryka, pomiędzy bramką, a kanałem tranzystora. Podatność dielektryka na uszkodzenie rośnie z każdym nowym procesem technologicznym, ponieważ zmniejsza się grubość izolatora bramki. Dodatkowo, nierównomierności występujące w nowych technologiach i bardzo cienkie warstwy (<10 nm) powodują szybkie zużywanie się dielektryka. Średni czas do uszkodzenia pod wpływem tego mechanizmu może być opisany równaniem [13]:

$$MTTF_{TDDB} \propto (\frac{1}{V})^{a-bT} e^{\frac{X+Y/T+ZT}{kT}}. \tag{2.8}$$

W równaniu (2.8) k jest stałą Boltzmanna, T oznacza temperaturę, V napięcie zasilania, a a, b, X, Y, Z są parametrami dopasowującymi do danych eksperymentalnych. Warto zauważyć, że $MTTF_{TDDB}$ zależy mocno od wartości napięcia zasilania.

Migracja naprężeniowa (ang. *Stress Migration – SM*), podobnie jak elektromigracja jest zjawiskiem powodującym przemieszczanie się atomów metalu w ścieżkach przewodzących. Jest spowodowana mechanicznym naprężeniem powstałym w skutek różnic w tempie rozszerzania się różnych materiałów pod wpływem zmiany temperatury. Zależność łącząca średni czas do uszkodzenia z temperaturą pod wpływem migracji naprężeniowej opisana może być jako [13]:

$$MTTF_{SM} \propto |T_0 - T|^{-n} e^{\frac{Ea_{SM}}{kT}}. \tag{2.9}$$

W zależności (2.9), n i Ea_{SM} są stałymi zależnymi od użytego materiału, a T_0 jest temperaturą w której napylana była warstwa metalizacji.

Cykle termiczne powodują zmęczenie materiału, a w konsekwencji jego uszkodzenie. Efekt ten jest w przypadku mikroprocesorów najbardziej widoczny w obudowach i złączach pomiędzy obudową a krzemem [12]. Średni czas do uszkodzenia zależy zarówno od wielkości jak i od częstotliwości występowania cykli termicznych. Przykładowo, w pracy [13] wykorzystano następujący model uszkodzeń spowodowanych cyklami termicznymi

$$MTTF_{TC} \propto \left(\frac{1}{T - T_{amb}}\right)^q, \tag{2.10}$$

gdzie q jest stałą określoną empirycznie.

W literaturze przedmiotu poświęca się wiele uwagi modelowaniu wpływu temperatury na niezawodność procesorów. Zagadnienia zużycia elementów mikroprocesora pod wpływem temperatury i dynamicznego zarządzania temperaturą podejmowane były między innymi w pracach [13, 14]. W pracy [15] badano wpływ temperatury na natężenie zjawiska elektromigracji w warstwie metalizacji w procesorze. W pracy [13] zaproponowano metodologię nazwaną `RAMP`, pozwalającą na symulowanie niezawodności procesorów na poziomie architektury komputerów. Wykorzystanie faktycznego obciążenia procesora zamiast najgorszego przypadku, przy jednoczesnym monitorowaniu zużycia, umożliwia obniżenie kosztów produkcji procesora. Zaproponowano również użycie tej metodologii do sterowania pracą procesora w celu zachowania zakładanego poziomu niezawodności przy zmiennym obciążeniu. Podobne podejście zaprezentowano w pracy [16]: ponieważ procesor nie pracuje w większości

przypadków cały czas w najgorszych warunkach pod względem temperatury, oraz napięcia zasilania, z czasem akumuluje „luz" lub „zapas" niezawodnościowy, który można wykorzystać w momentach zwiększonego zapotrzebowania na wydajność. Dzięki monitorowaniu pracy procesora, zaproponowany mechanizm pozwala na zwiększenie wydajności nawet o 20 – 30% przy zachowaniu określonego poziomu niezawodności. Metodologię `RAMP` uzupełniono i przystosowano do modelowania układów szeregowo-równoległych w pracy [17]. Na tej podstawie przeanalizowano możliwość zastosowania powielonych bloków funkcjonalnych do poprawienia niezawodności procesorów. W przypadku uszkodzenia bloku funkcjonalnego, nadmiarowy blok (dodany celowo, lub też tradycyjnie zwielokrotniany w procesorach superskalarnych) przejmuje jego zadania.

W pracy [18] przedstawiono metodologię symulacji niezawodnościowej układów SoC (ang. *System-on-Chip*), a w pracy [19] metody szeregowania zadań i zarządzania mocą w procesorach wielordzeniowych uwzględniające niezawodność. Wykorzystano model mechanizmów zużycia podobny do przedstawionego w pracy [13] aby pokazać, że mechanizmy których zastosowanie skutkuje podobną wydajnością, zużyciem energii, a nawet maksymalną temperaturą mogą mieć zauważalnie różny wpływ na niezawodność procesora.

2.2 Modelowanie termiczne procesorów

Pomiary temperatury działającego procesora wymagają zapewnienia niestandardowych warunków, na przykład chłodzenia materiałami przepuszczającymi promieniowanie podczerwone i nie są możliwe na wstępnym etapie jego projektowania. To z kolei ma wpływ na zachowanie termiczne układu [20]. Wynika stąd konieczność modelowania temperatury procesora, szczególnie na poziomie mikroarchitektury.

Temperatura procesora zależy od mocy rozpraszanej przez poszczególne jego komponenty. Jednym z pierwszych modeli opisujących zależność rozpraszanej przez procesor mocy od wykonywanych zadań, które były powszechnie wykorzystywane w pracach rozpatrujących temperaturę procesora na poziomie mikroarchitektury jest `Wattch` [21]. `Wattch` zawiera analityczne modele mocy rozpraszanej przez struktury zbudowane na bazie pamięci, w liniach dystrybucji sygnału zegarowego oraz przez logikę kombinacyjną. Symulator `Wattch` oblicza moc rozpraszaną w poszczególnych blokach

funkcjonalnych na podstawie danych o ich aktywności, które mogą pochodzić z symulatora funkcjonalnego takiego jak `gem5` [22] czy `SimpleScalar` [23], lub też z profili aktywności zebranych podczas działania fizycznego komputera. Działanie na wysokim poziomie abstrakcji pozwoliło na uzyskanie tysiąckrotnego przyspieszenia symulacji względem ówczesnych narzędzi operujących na kodzie HDL procesora (ang. *Hardware Description Language – język opisu sprzętu*), przy zachowaniu rozsądnego limitu błędu. Wattch był wykorzystywany w wielu pracach dotyczących zużycia energii przez procesory.

Symulatory mocy są wykorzystywane nie tylko w badaniach akademickich. W pracy [24] opisano symulator `ALPS` (ang. *Architectural-Level Power Simulator*) oraz proces projektowania procesora `Pentium 4` z jego wykorzystaniem. Często wykorzystywanym narzędziem jest również `CACTI` [25]. `CACTI` jest zintegrowanym modelem czasu dostępu oraz mocy statycznej i dynamicznej rozpraszanej przez pamięci podręczne. Na podstawie zadanej konfiguracji, `CACTI` oblicza szczegółowe statystyki pamięci podręcznych. Z uwagi na trudności z wydajnym dostarczaniem danych do procesora (ang. *memory wall*), znaczną część powierzchni współczesnych procesorów zajmują pamięci podręczne. Dlatego też uwzględnienie ich parametrów jest kluczowe dla dokładnego modelowania termicznego. Kolejnym wartym uwagi narzędziem jest `McPAT` [26] – zintegrowany model mocy, opóźnień i powierzchni procesorów wielordzeniowych. `McPAT` zawiera modele podstawowych elementów procesora wielordzeniowego – pamięci podręcznych, rdzeni, kontrolerów pamięci i drzew rozprowadzających sygnał zegarowy. W połączeniu z symulatorem funkcjonalnym pozwala on na przegląd przestrzeni możliwych rozwiązań z uwzględnieniem mocy rozpraszanej przez procesor.

`HotSpot` [27,28] jest modelem i symulatorem termicznym działającym na poziomie mikroarchitektury procesora. Na podstawie informacji o topografii układu (ang. *floorplan*), o mocy zużywanej przez poszczególne bloki funkcjonalne oraz o parametrach fizycznych procesora i obudowy, pozwala wyznaczyć przebieg temperatury w czasie w poszczególnych punktach procesora. Bardziej szczegółowy opis modelu `HotSpot` przedstawiono w rozdziale 3. Należy zaznaczyć, że w połączeniu z symulatorem funkcjonalnym i modelem mocy poszczególnych składowych procesora, `HotSpot` pozwala na przeprowadzenie realistycznych symulacji procesorów z uwzględnieniem temperatury. Analityczny model temperatury procesorów nazwany `ATMI` [29] pozwala na wydajną symulację termiczną działania procesorów wielordzeniowych. W odróżnieniu od `HotSpota` nie funkcjonuje on w dziedzinie dyskretnej, a wykorzystuje metody

analityczne i pewne uproszczające założenia co do procesora i jego obudowy.

W pracy [30] przeanalizowano zależności pomiędzy wydajnością, zużyciem energii i temperaturą procesorów wielordzeniowych. Wykorzystując rozbudowany zestaw symulatorów oraz wielowątkowe programy testowe zademonstrowano konieczność rozważania efektów związanych z temperaturą na poziomie mikroarchitektury. Z kolei w pracy [31] zaprezentowano wyniki badań nad wpływem mechanizmów dynamicznego zarządzania temperaturą na działanie procesorów wielordzeniowych. Pokazano, że różne parametry systemu mogą wymagać różnych metod zarządzania temperaturą. Na przykład w przypadku niskobudżetowych układów chłodzących (wysoka rezystancja termiczna) globalne skalowanie napięcia i częstotliwości w połączeniu z migracją wątków umożliwia uzyskanie większej wydajności niż metoda lepsza dla procesorów wyposażonych w wydajne chłodzenie: lokalne skalowanie napięcia i częstotliwości.

Wadą metod symulacyjnych w badaniach nad temperaturą procesorów jest długi czas obliczeń. Wynika on ze złożoności obliczeniowej samych algorytmów, ale również z konieczności uwzględnienia dynamiki procesów termicznych. Ponieważ zmiany temperatury procesora odbywają się w czasie od pojedynczych milisekund do setek sekund, symulacja termiczna i funkcjonalna procesora operują w różnych skalach czasowych. Liu i inni zaproponowali dwa szybkie algorytmy do symulacji termicznej pozwalające przyspieszyć obliczenia o 10 do 100 razy względem tradycyjnych metod przy minimalnej stracie dokładności. Algorytm `TMMSpectrum` [32] nadaje się szczególnie do krótkotrwałych symulacji termicznych, charakteryzujących się silnie okresowymi danymi wejściowymi. Analiza danych o rozpraszanej mocy w dziedzinie częstotliwości pozwala wyliczyć odpowiedź termiczną w stanie ustalonym. Profile mocy rozdzielane są na składową stałą i zmienną, skąd obliczana jest sumaryczna temperatura procesora. Z kolei, w pracy [33] zaproponowano metodę nazwaną `ThermSID` opartą na identyfikacji modelu stanu termicznego procesora. Na podstawie losowych pobudzeń budowana jest macierz Henkela parametrów Markowa, z której macierze stanu obliczane są metodą najmniejszych kwadratów.

W kolejnej pracy dotyczącej przyspieszania symulacji termicznych zaprezentowano narzędzie o nazwie `Temptor` [34] pozwalające śledzić temperaturę procesora w czasie działania na podstawie aktywności poszczególnych bloków funkcjonalnych z pomijalnym narzutem obliczeń.

Modelowaniu temperatury procesorów poświęcony jest rozdział 3, w którym przedstawione są wyniki pomiarów parametrów modelu termicznego procesora wielordze-

niowego oraz wysokopoziomowy model zachowania wątku przeznaczony do szybkich symulacji termicznych procesorów ogólnego przeznaczenia.

2.3 Dynamiczne zarządzanie temperaturą

Podczas wykonywania typowych zadań, moc rozpraszana przez procesor jest znacząco niższa niż maksymalna. Przekłada się to na dużo niższą temperaturę niż możliwa jest do osiągnięcia przy takiej samej temperaturze otoczenia, ale przy wyższym obciążeniu. Podobnie, w większości zastosowań procesory pracują w dużo lepszych warunkach termicznych niż wymagane do poprawnej pracy. Jednocześnie, koszt zapewnienia odporności procesora na bardzo wysokie obciążenia oraz pogorszone warunki otoczenia jest znaczący. Składają się na niego koszty zapewnienia odpowiedniej temperatury w projekcie mikroarchitektury, oraz koszty obudowy i układu chłodzenia. Dynamiczne zarządzanie temperaturą (ang. *Dynamic Temperature Management – DTM*) pozwala zmniejszyć koszty obudowy i układu chłodzenia w zamian za zmniejszenie wydajności zależne od obciążenia procesora.

Mechanizmy zarządzające temperaturą zyskują na znaczeniu szczególnie w przypadku nowych procesorów wielordzeniowych. W procesorze jednordzeniowym ograniczenie temperatury przekłada się bezpośrednio na konieczność ograniczenia mocy rozpraszanej. W procesorach wielordzeniowych, w większości przypadków, w naturalny sposób dostępne są niewykorzystane zasoby, które można wykorzystać bez ograniczania wydajności przetwarzania poszczególnych zadań, a temperatura rdzenia zależy nie tylko od jego obciążenia, ale również od zadań wykonywanych na pozostałych rdzeniach.

Mechanizmy DTM mają dwa powiązane ze sobą zadania. Pierwsze to niedopuszczenie do pracy powyżej ustalonej temperatury maksymalnej, co skutkuje obniżeniem niezawodności układu i może skutkować jego uszkodzeniem. Drugim celem jest maksymalizacja wydajności przetwarzania pomimo ograniczenia maksymalnej temperatury pracy.

2.3.1 Mechanizmy

Wysoka temperatura wynika z dużej gęstości mocy związanej z wykonywaniem dużej liczby operacji przez dany rdzeń procesora. Ideą działania mechanizmów DTM jest

rozproszenie obliczeń w czasie lub przestrzeni, co prowadzi do wyrównania gradientów i obniżenia maksymalnej temperatury. Rozproszenie obliczeń w czasie oznacza spowolnienie obliczeń. Może to być uzyskane przez obniżenie częstotliwości (i napięcia) pracy procesora, wstrzymywanie sygnału zegarowego lub spowalnianie wykonywania obliczeń przez mechanizmy działające na poziomie architektury procesora [5].

Dzięki możliwości ograniczenia napięcia zasilania wraz z częstotliwością zegara i idącymi za tym znacznymi oszczędnościami energii najczęściej używanym obecnie mechanizmem ograniczającym temperaturę jest dynamiczne skalowanie napięcia i częstotliwości (ang. *Dynamic Voltage and Frequency Scaling – DVFS*) [35]. Obniżenie częstotliwości pracy procesora skutkuje liniowym spowolnieniem obliczeń, a zmiana napięcia zasilania spadkiem energii proporcjonalnym do jego kwadratu. Ponadto, DVFS wykorzystywany jest do ograniczania zużycia energii w czasie, gdy pełna wydajność procesora nie jest potrzebna.

Alternatywą do spowolnienia wykonywania obliczeń jest rozproszenie ich w przestrzeni co pozwala zmniejszyć gęstość mocy i obniżyć maksymalną temperaturę. Na poziomie mikroarchitektury rdzenia procesora można to zrealizować np. przez rozproszenie lub zwielokrotnienie bloków funkcjonalnych. W procesorach wielordzeniowych, do wyrównania temperatury pomiędzy poszczególnymi rdzeniami można wykorzystać migrację wątków. Okresowe przenoszenie wątku, który powoduje rozpraszanie dużej mocy w rdzeniu na rdzeń o niższej temperaturze skutkuje obniżeniem maksymalnej temperatury. Przeniesienie wątku pomiędzy rdzeniami zajmuje pewien czas a dodatkowo powoduje zwykle chwilowe obniżenie wydajności związane z koniecznością ponownego wypełnienia pamięci podręcznych.

Zapewnienie wysokiej wydajności przy ograniczeniu temperatury nie jest zadaniem prostym i wymaga wiedzy o aktualnej temperaturze procesora, dynamice zmian temperatury oraz wpływie mechanizmów DTM na wydajność systemu. Dlatego też w literaturze znajduje się wiele publikacji na temat metod monitorowania i ograniczania temperatury procesorów [5].

2.3.2 Metody

Historycznie pierwsze, implementowane w procesorach ogólnego przeznaczenia, były proste mechanizmy DTM wstrzymujące działanie zegara procesora w momencie przekroczenia ustalonego progu temperatury. Jeśli współczynnik TDP układu chło-

dzącego jest wystarczająco wysoki, taki mechanizm wystarcza do zabezpieczenia układu w razie krytycznych sytuacji. Natomiast w warunkach niedostatecznego chłodzenia wstrzymywanie zegara może powodować utratę wydajności nawet o 87% w przypadku procesora Pentium 4 [36]. Ograniczenie spowolnienia wiązało się z koniecznością wykorzystania bardziej efektywnych mechanizmów, jak DVFS, oraz zastosowaniem lepszych, proaktywnych metod kontrolowania temperatury. Dla przykładu, przeniesienie wątku pomiędzy rdzeniami o znacząco różnej temperaturze pozwala na obniżenie maksymalnej temperatury. Jednak strata wydajności może być zauważalna i trudna do przewidzenia, więc mechanizm taki musi być stosowany z uwzględnieniem perspektywy odpowiednio długiego czasu.

Brooks i Martonosi [37] skupili się na ograniczeniu strat wydajności związanych z zastosowaniem metod DTM w procesorach wysokiej wydajności. Porównali również efektywność skalowania częstotliwości zegara oraz mechanizmów możliwych do zaimplementowania na poziomie mikroarchitektury procesora. Przykładowo, ograniczenie chwilowej mocy rozpraszanej przez procesor, przy małym spowolnieniu przetwarzania, można uzyskać ograniczając liczbę jednocześnie wykonywanych instrukcji. Innym sposobem jest wstrzymywanie spekulacyjnego wykonania rozkazów, do chwili rozwiązania konfliktu sterowania. Wstrzymanie dekodowania instrukcji pozwala na zakończenie wykonania już zdekodowanych instrukcji, ale wstrzymuje dekodowanie nowych co powoduje spadek temperatury z powodu niepełnego obciążenia potoku. Mechanizmy takie jak ograniczanie równoległości przetwarzania (ang. *Instruction-Level Parallelism – ILP*), wstrzymywanie dekodowania instrukcji, czy wstrzymywanie spekulacji pozwalają na uzyskanie jedynie liniowych oszczędności rozpraszanej mocy względem wydajności, lecz ich zaletą jest bardzo krótki czas aktywacji. W przypadku skalowania napięcia i częstotliwości procesora wynosi on około 10 μs.

Inny mechanizm DTM ściśle powiązany z mikroarchitekturą procesora został zaprezentowany w pracy [38]. W momencie przekroczenia dozwolonej temperatury, w przegrzewających się jednostkach wykonawczych obniżane jest napięcie zasilania. Ochronę przed uszkodzeniami typu opóźnienie zapewnia dwukrotne zmniejszenie częstotliwości taktowania tych jednostek, co nie przekłada się jednak na znaczące spowolnienie pracy całego potoku wykonawczego.

Ograniczenie negatywnego wpływu mechanizmów DTM na wydajność umożliwia również zastosowanie metod formalnych dotyczących różnych aspektów teorii sterowania. W pracy [39] opisano zastosowanie regulatora proporcjonalno-całkującego

PI co pozwoliło ograniczyć straty wydajności względem metod niewykorzystujących sprzężenia zwrotnego.

Połączenie właściwości technik poziomu mikroarchitektury oraz skalowania napięcia i częstotliwości pozwala zmniejszyć negatywny wpływ tych mechanizmów na wydajność. W pracy [40] omówiono technikę ograniczającą ILP w razie chwilowego podwyższenia temperatury oraz DVFS, który stosowany jest w sytuacjach, kiedy temperatura często przekracza dozwolony poziom. Odpowiednie sterowanie oboma mechanizmami pozwoliło na ograniczenie strat wydajności o 25%, względem wykorzystywania tylko jednego z nich. Z kolei w pracy [41] pokazano jak połączenie wstrzymywania sygnału zegarowego procesora oraz szeregowania zadań z uwzględnieniem temperatury pozwoliło na znaczącą redukcję spowolnienia przetwarzania zadań względem całkowicie sprzętowego mechanizmu.

Pojawienie się komercyjnie dostępnych procesorów wielowątkowych (ang. *Simultaneous MultiThreading — SMT*) spowodowało zainteresowanie nowymi możliwościami kontrolowania temperatury. W pracy [42] zaprezentowano metodę ograniczania temperatury w procesorze wielowątkowym, polegającą na modyfikacji algorytmu pobierania instrukcji. Jeśli dostępnych jest kilka wątków różniących się aktywnością procesora, w razie wykrycia wysokiej temperatury pobierane są instrukcje wątku, którego przetwarzanie ma najmniejszy wpływ na temperaturę. W porównaniu do algorytmu, który powoduje wstrzymywanie wykonania, taka metoda pozwala na poprawienie wydajności nawet o 30% oraz poprawę wskaźnika ED[21] o ponad 40%.

Moc rozpraszana przez poszczególne rdzenie w procesorach wielordzeniowych może się znacznie różnić. Przekłada się to na znaczące różnice w ich temperaturze, dlatego celowym jest stosowanie metod, które niezależnie kontrolują temperaturę każdego z rdzeni. Donald i Martonosi porównali 12 różnych metod kontrolowania temperatury w procesorach wielordzeniowych [43]. Najlepszy wynik uzyskali stosując skalowanie napięcia i częstotliwości każdego z rdzeni osobno w połączeniu z migracją wątków. Wstrzymywanie pracy całego procesora w przypadku lokalnego przegrzewania się skutkowało znacznym pogorszeniem wydajności. Z kolei, w pracy [44] zestawiono scentralizowane metody zarządzania temperaturą z rozproszoną kontrolą DVFS wykorzystującą sprzężenie zwrotne. Metody scentralizowane pozwalają potencjalnie

[1]Iloczyn energii i kwadratu opóźnienia (ang. *Energy-Delay*^{2}) jest miarą efektywności energetycznej. Jest często spotykany w literaturze przedmiotu do łącznej oceny wydajności i parametrów energetycznych procesorów.

na uzyskanie wyższej wydajności, można w nich również zaimplementować dodatkowe ograniczenia. Wśród wad scentralizowanego zarządzania pracą rdzeni należy wymienić wzrost złożoności obliczeniowej wraz ze zwiększeniem liczby rdzeni oraz spowodowane niedokładnością modelu ryzyko dużego odchylenia od optymalnych, a nawet zakładanych parametrów pracy – temperatury oraz wydajności. Dodatkowo, rozproszone sterowanie pozwala na łatwiejszą kompensację wariancji procesowej wpływającej na parametry poszczególnych rdzeni.

Istotnym zagadnieniem dotyczącym metod DTM jest zapewnienie poszczególnym zadaniom poziomu wydajności uwzględniającego ich priorytety. W pracy [45] zaproponowano algorytmy doboru częstotliwości rdzeni w procesorach wielordzeniowych w celu ochrony przed przekroczeniem maksymalnej zadanej temperatury. Przedstawiona metoda maksymalizuje wydajność z uwzględnieniem priorytetów poszczególnych zadań, oraz wpływu poszczególnych rdzeni na temperaturę ich sąsiadów. W pracach [46] oraz [47] na podstawie prostych modeli wydajności procesora wielordzeniowego wyszukiwane jest przyporządkowanie częstotliwości oraz napięć poszczególnych rdzeni optymalizujące wydajność przy ograniczeniu maksymalnej temperatury.

W pracy [48] zaprezentowano dwa algorytmy `HRTM` oraz `HRTA` (ang. *heat-and-run thread migration/assignment – migracja/przypisanie wątków wg zasady „podgrzej i uciekaj“*). Podstawą ich działania jest spostrzeżenie, że w procesorach z jednoczesną wielowątkowością (SMT) pracujących przy ograniczeniu temperaturowym, przetwarzanie mniejszej liczby wątków niż sumaryczna liczba sprzętowych kontekstów skutkuje większą sumaryczną wydajnością niż wypełnienie całego procesora. Nieefektywne wstrzymywania działania całego rdzenia procesora zastąpione zostało przyporządkowaniem zadań do rdzeni, a w chwili osiągnięcia wysokiej temperatury przeniesieniem ich kontekstów na na wolne miejsca w innych rdzeniach. Ponadto `HRTA` polega na przydzielaniu rdzeniom wątków komplementarnych pod względem wykorzystywanych zasobów co opóźnia moment przegrzania się procesora. Obie metody zostały zweryfikowane symulacyjnie na procesorze o czterech rdzeniach, z których każdy może wykonywać jednocześnie dwa wątki.

Metoda sterowania temperaturą procesora `HybDTM` [49] polega na połączeniu mechanizmów sprzętowych, na przykład wstrzymywania sygnału zegarowego, z mechanizmami programowymi takimi jak szeregowanie zadań uwzględniające temperaturę. Mechanizmy sprzętowe działają reaktywnie, jako zabezpieczenie w razie zagrożenia termicznego. Natomiast oprogramowanie, na podstawie modelu termicznego bazujące-

go na regresji, umożliwia prognozowanie wystąpienia wysokiej temperatury i wpływa na szeregowanie zadań.

Stochastyczny model kontrolowania temperatury procesora zaproponowano w pracy [50]. Sformułowano tam problem dynamicznego zarządzania temperaturą jako problem minimalizacji zużycia energii przy zakładanym poziomie wydajności i przy ograniczeniu maksymalnej temperatury jako kontrolowany proces decyzyjny Markowa. W pracy [51] podejmowane jest zagadnienie alarmowania o wysokiej temperaturze w obliczu niedokładnych i zaszumionych odczytów z czujników. Aby poradzić sobie z niepewnością dotyczącą prawdziwej temperatury złącza oraz rozpraszanej mocy, zastosowano filtr Kalmana oraz model procesów decyzyjnych Markowa. Pozwala to wiarygodnie określić temperaturę i prognozować wystąpienia gorących punktów.

Temperatura pracy procesora wpływa na prąd upływu, a więc praca w wysokiej temperaturze powoduje zużywanie dodatkowej energii. Zaprezentowana w pracy [52] łączna optymalizacja zużycia energii przez procesor i układ chłodzenia (wiatrak na radiatorze) pozwala na znaczące oszczędności całkowitej energii zużywanej przez komputer.

Aby uwzględnić skalowalność metod DTM, szybkość reakcji na zmiany temperatury oraz problemy związane z uruchamianiem wielu systemów operacyjnych na jednym komputerze (wirtualizacja), programowo-sprzętowa metoda opisana w pracy [53] wykorzystuje wirtualną warstwę oprogramowania posiadającego wiedzę o sprzęcie, na którym jest uruchomiona, do sterowania jego temperaturą i wydajnością.

Autorzy kilku prac [48,54–56] skupiają się na migracji wątków jako metodzie ograniczania temperatury procesorów wielordzeniowych. Metodę migracji wątków, która maksymalizuje wydajność przy ograniczonej temperaturze, minimalizując jednocześnie liczbę migracji i zapewniając sprawiedliwy przydział mocy, którą poszczególne wątki mogą rozproszyć zaproponowano w pracy [54]. Ważnym wnioskiem sformułowanym przez jej autorów jest fakt, że migracja wątków jest najbardziej skuteczna (umożliwia największy wzrost wydajności przy zadanym ograniczeniu na temperaturę maksymalną) w pierwszych kilkudziesięciu sekundach po obniżeniu liczby działających procesów.

Uwzględnienie temperatury procesora w szeregowaniu zadań w systemie operacyjnym pozwala ograniczyć liczbę sytuacji, w których maksymalna temperatura zostaje przekroczona. Autorzy pracy [57] zauważyli że kolejność szeregowania zadań różniących się rozpraszaną przez procesor mocą ma wpływ na jego temperaturę wynikową.

Zaszeregowanie na dwa następujące po sobie kwanty czasu dwóch programów sklasyfikowanych jako *gorący* i *zimny*, na podstawie temperatury procesora podczas ich nieprzerwanego wykonywania, najpierw *gorący*, a potem *zimny* da w efekcie niższą temperaturę niż w przypadku kolejności odwrotnej. Heurystyczny algorytm oparty na tym spostrzeżeniu, zaimplementowany w jądrze systemu `Linux`, pozwolił na zwiększenie wydajności o maksymalnie 7.6% przy niekorzystnych warunkach termicznych.

Mimo niewątpliwych zalet symulacji, cenne są prace, w których przeprowadzono eksperymenty na rzeczywistym sprzęcie. Brak weryfikacji założeń dotyczących właściwości fizycznych komputerów oraz wpływu systemu operacyjnego na temperaturę procesora może prowadzić do błędnych wniosków [3]. W pracy [58] zbadano różnice pomiędzy różnymi metodami ograniczania temperatury w rzeczywistym systemie z procesorem `POWER5`. Implementacja metod DTM w systemie operacyjnym pozwoliła na obniżenie maksymalnej temperatury o 5.5°C przy średnim ograniczeniu wydajności na poziomie 1%. Z kolei, w pracy [59] zaprezentowano wyniki uzyskane na podstawie implementacji szeregowania zadań uwzględniającego temperaturę w systemie `Linux`, uruchomionym na dwurdzeniowym procesorze. Ważne spostrzeżenie autorów dotyczy różnic wydajności oraz temperatury pomiędzy oboma rdzeniami procesora.

Kolejnym pomysłem na kontrolowanie temperatury procesora jest wykorzystanie sterowania predykcyjnego (ang. *Model Predictive Control – MPC*) [60, 61]. W pierwszej z wymienionych prac sformułowano problem sterowania pracą procesora przy zadanym ograniczeniu na maksymalną temperaturę oraz oczekiwaną wydajność. Zastosowanie sterowania predykcyjnego pozwala na uzyskanie *gładkiego* profilu temperatury. W praktyce oznacza to, że zamiast drastycznego ograniczania wydajności procesora w momencie przegrzewania się, jego szybkość jest stopniowo ograniczana. W każdej chwili zapewniony jest odpowiedni poziom wydajności, a jednocześnie temperatura nie przekracza wartości maksymalnej. W pracy [61] przedstawiono sposób na ograniczenie złożoności klasycznych regulatorów wykorzystujących sterowanie predykcyjne do ograniczania temperatury w procesorach wielordzeniowych. Zaprezentowane podejście polega na rozproszeniu sterowania pomiędzy rdzenie. Każdy rdzeń obsługiwany jest przez niezależny regulator. Każdemu z rdzeni przyporządkowana jest częstotliwość, która pozwala zminimalizować energię oraz różnicę względem oczekiwanej wydajności przy jednoczesnym ograniczeniu temperatury. Wymiana informacji pomiędzy regulatorami sąsiednich rdzeni pozwala zwiększyć ogólną wydajność systemu.

Odmienne od wyżej wymienionych, wykorzystujących skalowanie napięcia i częstotliwości, podejście do ograniczania temperatury procesora zaprezentowano w pracy [62]. Metoda nazwana `Dimetrodon` działa na poziomie algorytmu szeregowania zadań w systemie operacyjnym i polega na wstrzykiwaniu bezczynnych cykli w miejsce aktywnych programów, aby obniżyć temperaturę rdzenia. Algorytm szeregowania zadań wstrzymuje proces, aby nie został uruchomiony na innym rdzeniu i zamiast niego przydziela czas bezczynnemu procesowi. W porównaniu do DVFS, metoda ta jest mało efektywna w długich okresach, natomiast pozwala na drobnoziarniste sterowanie temperaturą. Jest ponadto niezależna od sprzętu i pozwala na uwzględnienie priorytetów zadań oraz informacji o wzorcach i częstotliwości wywłaszczeń procesów.

Na podstawie obszernej literatury przedmiotu można podzielić mechanizmy dynamicznego zarządzania temperaturą według następujących kryteriów:

- reaktywne, które chronią procesor przed uszkodzeniem ograniczając wydajność w momencie wykrycia przekroczenia maksymalnej dopuszczonej temperatury i predykcyjne, które podejmują działanie zawczasu na podstawie prognozy podwyższonej temperatury, np. [63–67],

- działające globalnie – spowalniając pracę całego procesora, lokalnie w obrębie pojedynczego rdzenia i lokalne na poziomie poszczególnych bloków funkcjonalnych rdzenia,

- bazujące na spowolnieniu pracy, na przykład poprzez bramkowanie sygnału zegarowego, (*clock gating*) czy skalowanie napięcia i częstotliwości [45, 47, 68, 69] oraz opierające się na migracji wątków do innych rdzeni, migracji obliczeń do zapasowych bloków funkcjonalnych lub szeregowaniu zadań uwzględniającym temperaturę [54–56, 70],

- heurystyczne i oparte na metodach formalnych teorii sterowania, w tym wykorzystujące regulatory PID [39] oraz sterowanie predykcyjne [60, 61, 71],

- programowe, sprzętowe i programowo-sprzętowe [72],

- wykorzystujące pojedynczą technikę, np. dynamiczne skalowanie częstotliwości (ang. *Dynamic Frequency Scaling – DFS*), DVFS, migracja wątków (ang. *Thread Migration – TM*), wstrzymywanie pobierania instrukcji (ang. *fetch throttling*), lub synergiczne połączenie wielu technik [41],

- specjalizowane do aplikacji multimedialnych, wykorzystujące fakt, że aplikacje takie zwykle wykazują cykliczne zachowanie, np. dekodując kolejne klatki, lub grupy klatek filmu i dzięki temu możliwa jest prognoza temperatury oraz dostosowanie wydajności procesora do wymaganego czasu dekodowania [73, 74],

- uwzględniające specyfikę urządzeń wbudowanych [75] oraz zaprojektowane z myślą o procesorach ogólnego przeznaczenia.

2.3.3 Trendy

Prognozowany na najbliższe lata [76] nieprzerwany postęp w dziedzinie wytwarzania układów wysokiej skali integracji (VLSI) umożliwi wykorzystanie setek miliardów tranzystorów w pojedynczym procesorze. Niestety, zgodnie z regułą Pollacka[2] wydajność pojedynczego rdzenia procesora jest proporcjonalna do pierwiastka powierzchni przez niego zajmowanej. Z tego powodu efektywnym sposobem wykorzystania ogromnych zasobów logiki jest produkcja procesorów z bardzo dużą liczbą rdzeni, sięgających setek, a nawet tysięcy [77]. Zapewnienie efektywnego przetwarzania, dostosowanie chwilowej wydajności do zapotrzebowania na moc obliczeniową oraz zapewnienie pracy takich procesorów w odpowiedniej temperaturze stanowi wyzwanie, z którym producenci będą się musieli zmierzyć już wkrótce. Sztandarowym przykładem nadchodzących zmian jest procesor `SCCC` (Single-Chip Cloud Computer), zawierający 48 rdzeni połączonych zintegrowaną siecią [78]. Każda z par rdzeni zawiera zintegrowany czujnik temperatury i zapewnia możliwość kontrolowania częstotliwości swojej pracy. Dodatkowo cały procesor podzielony jest na sześć odrębnych domen z niezależnym zasilaniem (ang. *voltage islands*), co umożliwia kontrolę temperatury oraz precyzyjne sterowanie wydajnością i zużyciem energii.

Jednym z pomysłów na zwiększenie wydajności i efektywności energetycznej, zarówno podczas przetwarzania zadań szeregowych jak i równoległych, jest zastosowanie procesorów heterogenicznych. Dostosowanie złożoności wykorzystywanego rdzenia do aktualnie przetwarzanego zadania pozwala na ograniczenie zużycia energii przy odpowiednim poziomie wydajności. Minusem stosowania rdzeni o różnej wydajności i złożoności jest występowanie dużych różnic temperatury pomiędzy nimi. W pracy [79] autorzy opisują metodę minimalizującą zużycie energii w procesorze heterogenicznym,

[2]Termin pochodzi od Freda Pollacka inżyniera w firmie `Intel`.

przy jednoczesnym zapewnieniu równomiernej temperatury całego procesora. Ciekawym przykładem procesora heterogenicznego jest `Cell BE` [80], w którym połączono rdzeń `Power PC` oraz 8 specjalizowanych jednostek o nazwie `SPE` (ang. *Synergistic Processing Element*), operujących jedynie na lokalnej pamięci podręcznej. Za cenę skomplikowanego modelu programowego, otrzymano procesor charakteryzujący się dużą efektywnością energetyczną.

Postępy technologiczne umożliwiają produkcję układów składających się ze zintegrowanych warstw, komunikujących się poprzez połączenia poprowadzone przez podłoża poszczególnych warstw (ang. *Through-Silicon Via – TSV*). Mówi się tu zwykle o integracji 3D – w trzecim wymiarze. Układy wykonane w takich technologiach umożliwiają zwiększenie skali integracji, lecz pojawia się problem odprowadzania ciepła. Mimo, że wielowarstwowe układy 3D nie są jeszcze masowo produkowane, istnieje już sporo publikacji podejmujących temat dynamicznego zarządzania temperaturą w procesorach 3D [2] oraz planowania rozłożenia bloków funkcjonalnych (ang. *floorplaning*) w procesorach 3D z uwzględnieniem temperatury [81].

Coraz częściej w komercyjnych procesorach ogólnego przeznaczenia pojawiają się mechanizmy zarządzania temperaturą, takie jak technologia `Turbo Boost` [10, 11], oraz czujniki temperatury w każdym rdzeniu procesora. Widać też coraz większe zainteresowanie zagadnieniami temperatury w procesorach i jej wpływu na wydajność ze strony osób zajmujących się algorytmami stosowanymi w systemach operacyjnych [82]. Można się spodziewać, że w celu uzyskania maksymalnej wydajności i efektywności przetwarzania, zagadnienia związane z temperaturą procesorów będą widoczne w wyższych warstwach – systemach operacyjnych i językach programowania.

Ziarnistość

Skalowanie procesów technologicznych powoduje zwiększanie liczby rdzeni w pojedynczym układzie i jednocześnie zmniejszanie ich wymiarów liniowych. Ponieważ procesor zachowuje się jak przestrzenny filtr dolnoprzepustowy dla temperatury, zmniejszanie wymiarów poszczególnych rdzeni spowoduje obniżenie się temperatury maksymalnej przy stałym współczynniku TDP. W pracy [83] przedstawiono analityczny model określający moc maksymalną w homogenicznych procesorach wielordzeniowych. Obniżenie mocy maksymalnej nastąpi przy założeniu, że poszczególne rdzenie są oddzielone obszarami relatywnie chłodniejszych pamięci podręcznych.

Geometria i wymiary poszczególnych rdzeni i bloków funkcjonalnych mają duży wpływ na temperaturę jaką mogą osiągnąć oraz na poziom złożoności na jakim należy monitorować i ograniczać temperaturę – bloków funkcjonalnych, rdzeni procesora lub grup rdzeni. Jednym z tematów podejmowanych w pracy [84] była ocena czy możliwe jest wygenerowanie gorącego punktu w linii pamięci podręcznej przez bardzo częste odwoływanie się do niej. Taki efekt mógłby stanowić podstawę ataku na procesor, mającego na celu jego termiczne uszkodzenie. Okazuje się, że na temperaturę osiąganą przez pojedynczą linię pamięci ma wpływ jej implementacja, a konkretnie stosunek wymiarów liniowych.

2.4 Prognoza temperatury w procesorach wielordzeniowych

W pracy [67] zaprezentowano mechanizm dynamicznego zarządzania temperaturą procesorów wielordzeniowych wykorzystujący prognozę temperatury. Na podstawie przedstawionych eksperymentów symulacyjnych autorzy ocenili, że zaprezentowana metoda pozwala na zmniejszenie średniej i ograniczenie maksymalnej temperatury procesora, przy jednoczesnym zwiększeniu wydajności pracy. Prostą metodę wykorzystującą regresję do przewidywania gorących punktów podano w pracy [66]. Temperatura przewidywana jest na podstawie równania liniowego $Y = aX + b$, gdzie X to wartość licznika zdarzeń dla danego bloku funkcjonalnego, a a i b to stałe określone podczas projektowania systemu. W połączeniu z DVFS, zaprezentowana metoda pozwala na uniknięcie większości zagrożeń spowodowanych wysoką temperaturą przy minimalnym narzucie koniecznym na obliczenia prognozy. W pracy [85] przedstawiono prognozę temperatury na podstawie poprzednio odczytanych wartości stosując model autoregresywnej średniej kroczącej (ang. *autoregressive moving average – ARMA*). Dwie podstawowe wady tej metody to założenie o tym, że modelowany proces stochastyczny jest stacjonarny oraz konieczność identyfikacji parametrów modelu. W przypadku często zmieniającego się obciążenia pozostałych rdzeni procesora lub naprzemiennego wykonywania więcej niż jednego programu na tym samym rdzeniu, może się okazać, że przebieg temperatury danego rdzenia ma charakter niestacjonarny i algorytm wykorzystujący ARMA poświęci większość czasu na aktualizację parametrów modelu.

Zarówno autorzy pracy [85], jak i [67] weryfikują mechanizm predykcji na wycinkowych danych o temperaturze jednego rdzenia. Zaprezentowane przykłady można opisać jako procesy stacjonarne. Jednakże, jak zostanie to pokazane w rozdziale 4 w praktyce istotne zmiany temperatury procesora mają najczęściej związek z uruchamianiem i zakańczaniem zadań (programów, wątków) i możliwość prognozy temperatury na podstawie jej poprzednich odczytów jest w takich sytuacjach mocno ograniczona.

W pracy [64] przedstawiono porównanie różnych metod prognozy temperatury procesora wielordzeniowego. Oprócz metody opartej na modelu ARMA, przeanalizowano prognozę na podstawie poprzednich wartości temperatury. Działa ona wykorzystując tablicę podobną do układu globalnej prognozy rozgałęzień, składającą się z rejestru przesuwnego przechowującego kilka ostatnich odczytów i tablicy indeksowanej zawartością tego rejestru. Tablica przechowuje ciągi kolejnych wartości temperatury wraz z odpowiadającymi im prognozami. W pracy [64] analizowane są również metody prognozy wykorzystujące średnią wykładniczą (ang. *exponential moving average*) oraz rekursywną metodę najmniejszych kwadratów (ang. *Recursive Least Square Method*) [63].

W pracy [53] temperatura prognozowana jest na podstawie n ostatnich odczytów. Do wyliczenia, kiedy temperatura przekroczy ustaloną wartość progową, wykorzystywane jest równanie:

$$T_0 + \Delta T \cdot x = T_{th}, \tag{2.11}$$

gdzie T_0 jest aktualną temperaturą, ΔT jest różnicą temperatur pomiędzy dwoma sąsiednimi odczytami, T_{th} jest temperaturą progową. Obliczona wartość x oznacza liczbę okresów po których spodziewane jest przekroczenie temperatury.

Autorzy pracy [86] zaproponowali algorytm pozwalający na dokładną prognozę temperatury procesorów wielordzeniowych na podstawie globalnie zdefiniowanych *faz* wykonania programów. W tym kontekście faza jest etapem wykonania programu, który charakteryzuje się jednorodnymi parametrami wydajności, mocy i temperatury. Algorytm działający w fazie uczenia się (ang. *off-line*) wiąże temperaturę z częstotliwością pracy i fazą programu. Prognoza temperatury w kroku $k+1$ odbywa się na podstawie aktualnej fazy p oraz częstotliwości zegara f zgodnie z równaniem:

$$T_i[k+1] = \sum_{j=1}^{N} a_{ij} T_j[k] + b_i(f, p), \tag{2.12}$$

gdzie, i oraz j są indeksami rdzeni, a parametry a oraz b wyznaczone są eksperymentalnie. Wartości współczynników $a_{i,j}$ wyznaczane są metodą najmniejszych kwadratów na podstawie obserwacji zmiany temperatury nieobciążonych rdzeni znajdujących się w sąsiedztwie obciążonego rdzenia. Zaprezentowana metoda została zweryfikowana eksperymentalnie na komputerze z czterordzeniowym procesorem Intel Core i7. Z kolei, w pracy [65] temperatura i-tego rdzenia w kroku $k+1$ prognozowana jest na podstawie równania:

$$t_{k+1,i} = t_{k,i} + \sum_{\forall j \in Adj_i} a_{i,j}(t_{k,j} - t_{k,i}) + b_i p_i, \tag{2.13}$$

gdzie współczynniki $a_{i,j}$ i b_i są określone na podstawie właściwości modelu termicznego. Z uwagi na stabilność numeryczną równanie 2.13 musiało być rozwiązywane z krokiem 0.4 ms. Żeby uzyskać okres prognozy 100 ms, potrzeba aż 250 iteracji. Równanie (2.13) jest równoważne równaniu (2.12), jednak autorzy pracy [86] nie wspominają o problemach ze stabilnością numeryczną i wykonują obliczenia co 100 ms.

W pracy [63] zaproponowano model prognozy temperatury łączący prognozę temperatury rdzenia i prognozę temperatury aplikacji. Prognoza temperatury aplikacji odbywa się na podstawie dotychczasowego przebiegu temperatury z wykorzystaniem rekursywnej metody najmniejszych kwadratów.

2.5 Szacowanie wydajności procesorów pod ograniczeniem temperatury

Regularna i symetryczna struktura procesorów wielordzeniowych pozwala na analityczne podejście do budowy modeli łączących ich wydajność, rozpraszaną moc oraz temperaturę. Zastosowanie prostych, homogenicznych rdzeni w procesorach wykorzystujących sieci jednoukładowe (ang. *Network-on-Chip*) np. [78] przemawia za możliwością wykorzystania wysokopoziomowych modeli do oceny ich wydajności na wczesnym etapie projektowania.

Rao i inni [55] zaprezentowali analityczną metodę przeglądu przestrzeni możliwych rozwiązań dla procesorów wielordzeniowych, uwzględniającą migrację wątków stosowaną do ograniczania temperatury. Prognozowana w tej pracy wydajność procesora obliczona analitycznie zgadza się z wynikami symulacji numerycznych z maksymalnym błędem na poziomie 5%. Jednocześnie podejście analityczne pozwala na przy-

spieszenie oceny rozwiązania o sześć rzędów wielkości. W pracy [87] wyprowadzone są wyrażenia opisujące wydajność procesora wielordzeniowego w stanie ustalonym przy ograniczeniu temperaturowym. Ich weryfikację przeprowadzono względem termicznego symulatora systemowego, uwzględniającego proste metody DTM. W obu pracach: [55] oraz [87] przyjęto bardzo prosty model wątku, w którym wydajność i moc rozpraszana podczas jego wykonywania zależą jedynie od częstotliwości pracy rdzenia oraz jego napięcia zasilania. Przy zadanej szybkości pracy rdzenia, jego zużycie energii jest stałe w czasie i jednakowe pomiędzy wątkami. Taki sam model wątku wykorzystano w pracach [56] oraz [68], w których zaprezentowano optymalne pod względem przepustowości systemu metody migracji wątków i sterowania DVFS w procesorach ograniczonych temperaturowo.

W pracy [88] przedstawiono analizę wpływu temperatury na wydajność procesorów wielordzeniowych. Zaprezentowano wysokopoziomowy, analityczny model wydajności procesorów wielordzeniowych, wykorzystujący jednak bardzo uproszczony model zadań oparty na prawie Amdahla. Uwzględnienie temperatury sprawia, że procesory asymetryczne, w których jeden wysokowydajny rdzeń przeznaczony jest do szybkiego przetwarzania szeregowych zadań, traci na atrakcyjności. Jest tak, ponieważ rdzeń procesora o dużej złożoności jest prawdopodobnym miejscem, w którym procesor będzie się przegrzewał.

Za stosowaniem podejścia analitycznego do oceny właściwości termicznych architektur wielordzeniowych przemawia wymagany bardzo krótki czas oceny danego rozwiązania. Dodatkowo, wraz z nieuchronnym zwiększaniem się liczby rdzeni w procesorach [77], sumaryczna wydajność procesora będzie coraz bardziej uzależniona od jego parametrów termicznych, a coraz mniejszy wpływ na wydajność będą miały zdarzenia w poszczególnych rdzeniach. Z drugiej strony, w procesorach wysokorównoległych (ang. *many-core*), współzależności w dostępie do współdzielonych zasobów (np. pamięci podręcznych, kontrolerów i magistral pamięci) będą miały coraz większe znaczenie dla szybkości przetwarzania poszczególnych wątków [89]. Dodatkowo, specyfika programów wielowątkowych, które wraz z rozwojem architektur wielordzeniowych będą coraz powszechniejsze, sprawia, że proste oszacowania parametrów całego systemu komputerowego mogą być dalekie od rzeczywistych wyników. Konieczność synchronizacji w programach wielowątkowych wprowadza skomplikowane zależności pomiędzy wydajnością przetwarzania poszczególnych wątków, a wydajnością całego systemu [90].

2.6 Konkluzje

W analizie literatury przedmiotu pominięto szereg prac dotyczących pokrewnych tematów takich jak rozmieszenie bloków funkcjonalnych (ang. *floorplaning*) z uwzględnieniem temperatury [91], pomiary temperatury przez materiał termoprzewodzący [20], weryfikacja mocy rozpraszanej przez procesory poprzez pomiar temperatury [92], czy też odczytywanie mocy na podstawie temperatury [93].

Podobnie pominięta została bogata literatura na temat przetwarzania przy ograniczeniu maksymalnej mocy oraz minimalizacji energii podczas obliczeń. Choć wykorzystuje się tu mechanizmy takie jak w zarządzaniu temperaturą, to funkcje celu są w obu przypadkach znacząco różne.

Jakkolwiek badanie różnych aspektów metod DTM ma walor naukowo-poznawczy, to brakuje standardów i dobrych praktyk w temacie oceny poszczególnych rozwiązań oraz weryfikacji wyników uzyskanych przez innych autorów. Typowe jest przyjmowanie nierealistycznych założeń dotyczących właściwości analizowanych systemów. W efekcie utrudnione jest porównanie wyników prac w temacie dynamicznego zarządzania temperaturą, ponieważ wyniki poszczególnych eksperymentów zależą mocno od warunków otoczenia, programów oraz sprzętu na jakim wykonywano badania, a także szczegółów implementacji.

Z powodu zmian mocy rozpraszanej podczas wykonywania różnych programów, temperatura otoczenia w stanie ustalonym może się znacząco różnić między nimi. Dotyczy to zwłaszcza komputerów przenośnych, których systemy chłodzenia mogą być mniej wydajne niż w przypadku serwerów. Autorzy pracy [94] postulują uzależnienie zakładanej w symulacjach termicznych temperatury otoczenia od aktualnie przetwarzanego programu. Przyjęcie stałej temperatury otoczenia może prowadzić do błędnych wniosków. W procesorach (systemach) z ograniczeniem temperatury, do oceny wydajności należy brać też pod uwagę warunki otoczenia i właściwości układu chłodzącego.

Niestety, wielu autorów ogranicza się w swoich pracach do podejścia analitycznego lub symulacyjnego, zaś dobór parametrów symulacji jest często dyskusyjny. Z kolei wadą wykonywania eksperymentów na fizycznych maszynach jest brak elastyczności – brakuje możliwości zmiany parametrów systemu, na którym wykonuje się badania. Wciąż brak spójnych i szeroko akceptowanych metod oceny poszczególnych rozwiązań. Niestety, w wielu pracach wykorzystano jedynie symulacje do oceny skuteczno-

ści mechanizmów DTM. Z uwagi na znaczny czas symulacji, eksperymenty obejmują często zbyt krótki okres, żeby uwzględnić zachowanie termiczne radiatora, co może znacząco zaburzać ich wyniki. Częstym błędem popełnianym przez autorów jest podawanie ograniczenia temperatury w procentach. Na przykład w pracy [38] autorzy podają, że ich metoda pozwala na ograniczenie temperatury o 6,6% w średnim przypadku. Interpretacja tego wyniku pozostawiona jest już czytelnikom.

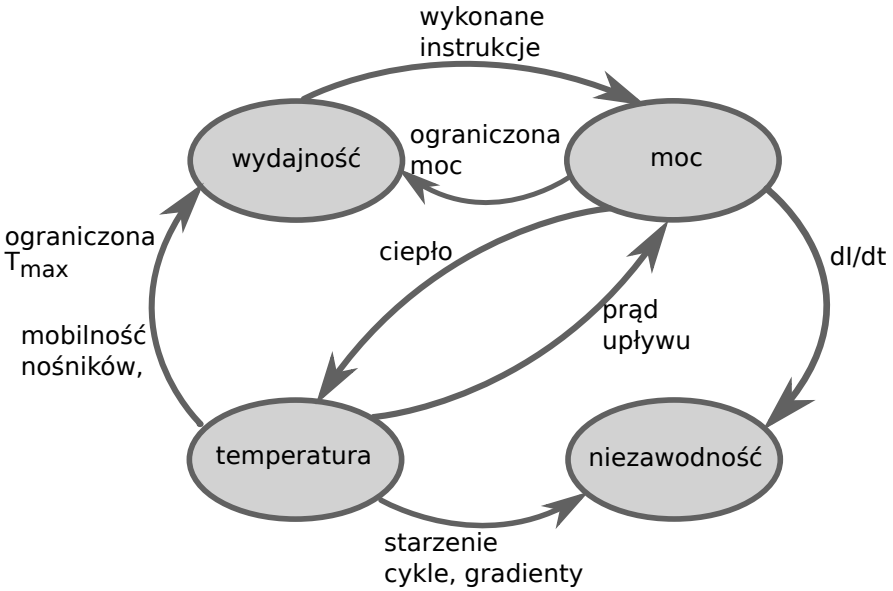
Bogata literatura na temat dynamicznego zarządzania temperaturą w procesorach świadczy o ważności zagadnienia. Nie oznacza to jednak wcale, że problemy związane z temperaturą w procesorach wielordzeniowych są już rozwiązane.

Rozdział 3

Modelowanie termiczne procesorów

Potrzeba uwzględnienia wpływu temperatury na działanie współczesnych procesorów występuje na wszystkich etapach ich projektowania (rys. 3.1). Temperatura ogranicza maksymalną częstotliwość taktowania układu z powodu negatywnego wpływu na mobilność nośników prądu oraz rezystancję w połączeniach [6]. Prąd upływu jest wykładniczo zależny od temperatury. Ponadto wzrost temperatury procesora ma bezpośredni negatywny wpływ na niezawodność układu. Procesy takie jak elektromigracja, czy starzeniowe przebicie dielektryka nasilają się wykładniczo wraz ze wzrostem temperatury, jak opisano to w rozdziale 2.1.1. Ponieważ temperatura pracy wpływa na niezawodność i rozpraszaną moc konieczne jest jej modelowanie już na wczesnym etapie projektowania procesora. Odbywa się to zwykle na poziomie abstrakcji odpowiadającym poszczególnym blokom funkcjonalnym procesora [24]. Ponadto, po wyprodukowaniu próbnych egzemplaży procesora, konieczna jest weryfikacja zachowania termicznego oraz ilości rozpraszanej energii [92].

Uwzględnienie temperatury w procesie projektowania procesora wymaga uzyskania danych na temat zachowania programów (rys. 3.2). Na podstawie informacji o dynamicznym przebiegu programów budowane są informacje o wzorcach pobudzeń poszczególnych elementów procesora. Wykorzystywane są do tego pomiary działania już istniejących procesorów oraz symulatory funkcjonalne, takie jak SimpleScalar [23] czy gem5 [22]. Uzyskane wzorce wykorzystywane są w symulatorach mocy rozpraszanej przez układ. Większość modeli mocy stosowanych w badaniach nad architekturą komputerów stanowi połączenie analitycznych zależności wiążących pobudzenia ze zużywaną energią oraz empirycznych danych dostępnych z pomiarów już wyprodukowanych układów. Typowymi przykładami są tu CACTI [25] – model pamięci pod-

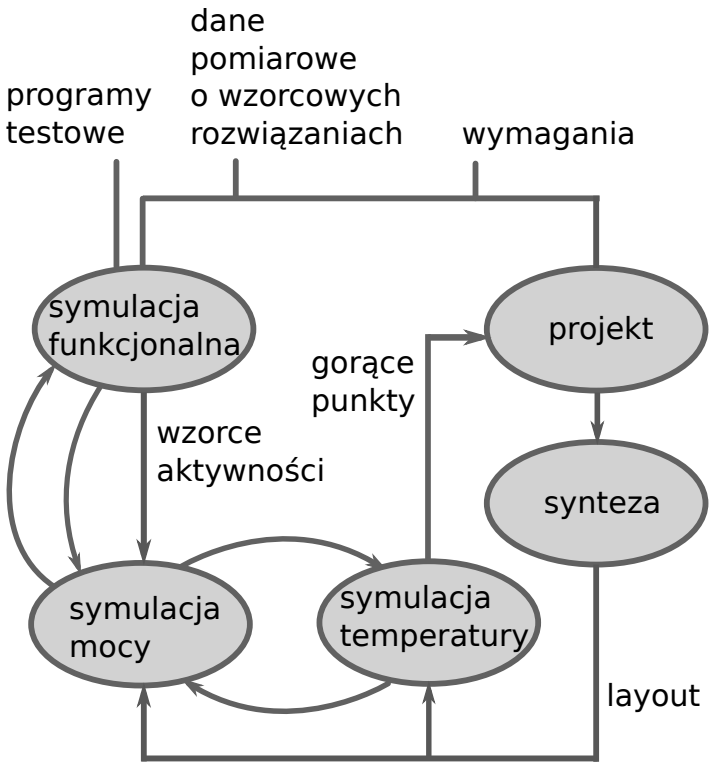


Rysunek 3.1: Zależność wydajności procesorów wielordzeniowych, rozpraszanej mocy, temperatury oraz niezawodności

ręcznych, Wattch [21] oraz McPAT [26] – modele mocy całego procesora. Z uwagi na dodatnie sprzężenie zwrotne pomiędzy mocą i temperaturą procesora, konieczne jest również, podczas modelowania nowych układów, wykorzystanie modeli termicznych. Na wczesnym etapie projektowania układu, kiedy szczegóły implementacji nie są jeszcze znane, dokonywany jest zwykle przegląd przestrzeni możliwych rozwiązań – analizowany jest wpływ poszczególnch parametrów na wydajność układu. Na tym etapie symulacji ważna jest jej wydajność, nawet kosztem bezwzględnej dokładności. Duża liczba istotnych parametrów opisujących architekturę procesora wielordzeniowego oraz wykładniczo rosnąca liczba ich komibnacji powodują, że potrzebne są abstrakcyjne, wysokopoziomowe modele pozwalające w rozsądnym czasie znaleźć potencjalnie atrakcyjne konfiguracje [95]. Nie jest to zadaniem łatwym z uwagi na nieliniowe i nieoczywiste zależności pomiędzy wydajnością, zużyciem energii i temperaturą.

Podczas projektowania, dopiero po ustaleniu założeń architektury procesora, potrzebne są dokładne modele mocy i temperatury odzwierciedlające szczegóły mikroarchitektury aktualnego projektu.

Postęp w dziedzinie wytwarzania układów wielkiej skali integracji (ang. *VLSI*) połączony z rozwojem architektur procesorów wielordzeniowych spowodował zmianę poziomu abstrakcji modelowania termicznego procesorów. Z uwagi na malejące rozmia-

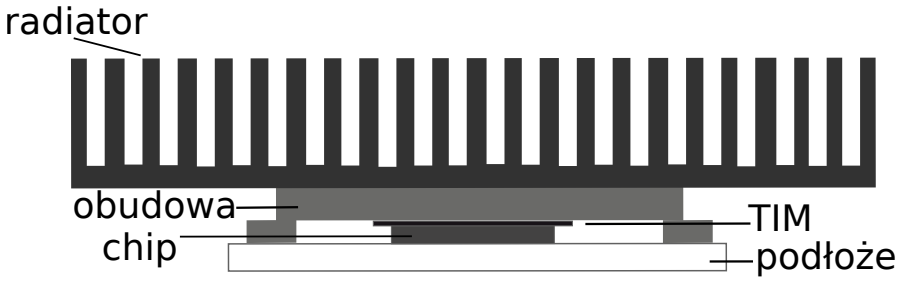


Rysunek 3.2: Modelowanie temperatury procesora na różnych etapach jego projektu

ry poszczególnych bloków funkcjonalnych, coraz częściej podstawowym elementem, któremu przyporządkowane są położenie, zużycie energii oraz temperatura, jest cały rdzeń procesora. Konsekwencje tej zmiany z punktu widzenia modelowania termicznego są dwojakie. Model procesora zbudowanego jako regularna struktura składająca się z jednakowych rdzeni (np. [78]) jest mniej złożony niż termiczny model rdzenia zbudowanego z poszczególnych bloków funkcjonalnych. Z uwagi na regularną strukturę możliwe jest również podejście analityczne do szacowania wydajności takiego układu [87]. Z drugiej strony dokładna symulacja zachowania procesora wielordzeniowego wymaga symulacji całego systemu operacyjnego i jednoczesnego wykonywania wielu aplikacji. Implikuje to długi czas takich symulacji.

3.1 Model termiczny procesora wielordzeniowego

Na rysunku 3.3 pokazano schematyczny przekrój przez współczesny procesor wysokiej wydajności wraz z obudową i radiatorem. Chip połączony jest z obudową materiałem termoprzewodzącym (ang. *Thermal Interface Material – TIM*). Oprócz ochrony

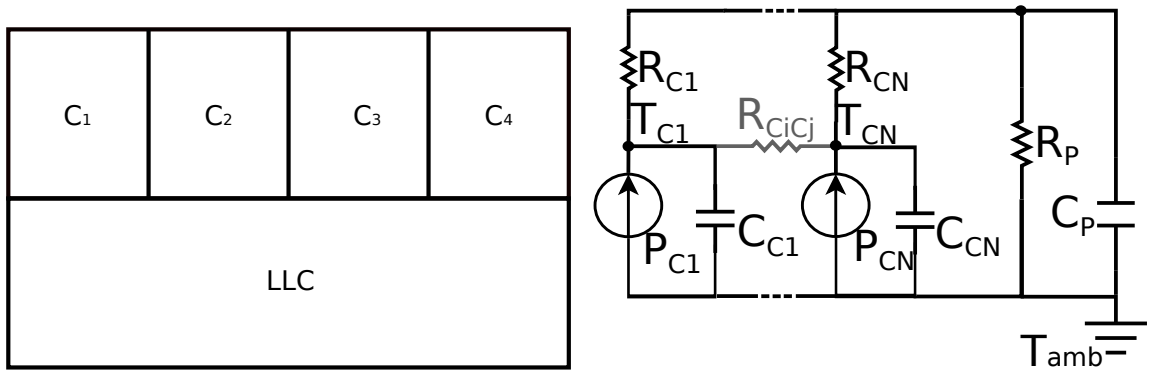


Rysunek 3.3: Schematyczny przekrój przez procesor, obudowę oraz radiator współczesnego procesora w komputerze stacjonarnym

przed mechanicznym uszkodzeniem, głównym zadaniem obudowy jest odprowadzenie ciepła do radiatora. Z tego powodu ma ona znacząco większą powierzchnię niż sam procesor. Modelowanie temperatury procesora wymaga również uwzględnienia parametrów używanego chłodzenia – w większości przypadków jest to radiator, charakteryzujący się masą oraz wymiarami znacznie większymi od obudowy procesora, oraz wiatrak wymuszający przepływ powietrza przez grzebienie radiatora. Prędkość obrotowa wiatraka ma znaczący wpływ na rezystancję termiczną całego układu chłodzenia.

Najczęściej używane termiczne modele poziomu mikroarchitektury procesora opierają się na analogii między zjawiskami elektrycznymi a cieplnymi [7,96]. W takich modelach temperatura odpowiada napięciu, ciepło natężeniu prądu elektrycznego, a źródła ciepła idealnym źródłom prądowym. Każdemu blokowi procesora (rdzenie i ich elementy składowe, pamięci podręczne) odpowiada jeden węzeł termicznego obwodu RC. Sztandarowym przykładem takiego modelu jest HotSpot [20, 27, 28] udostępniany jako rozbudowane narzędzie do termicznej symulacji procesorów. Na podstawie konfiguracji (wymiary procesora i obudowy, rezystancja termiczna radiatora, topografia układu (ang. *floorplan*)) oraz informacji o mocy rozpraszanej przez procesor, program oblicza przebieg temperatury w poszczególnych punktach układu. HotSpot jest udostępniany jako program o otwartym kodzie źródłowym, co sprzyja integracji z symulatorami architektury komputerów. HotSpot był wykorzystywany przy projektach zakończonych wieloma publikacjami, a jego kod był sprawdzany przez dużą liczbę użytkowników [3].

Rysunek 3.4a przedstawia schemat przykładowego procesora zbudowanego z 4 rdzeni (C_1 do C_4) umieszczonych w linii oraz dużego obszaru wspólnej pamięci podręcznej



a) b)

Rysunek 3.4: Przykładowa topografia i ogólny termiczny model RC procesora wielordzeniowego

ostatniego poziomu (ang. *Last Level Cache – LLC*) umieszczonego równolegle do rdzeni. Rys. 3.4b zawiera schemat termicznego obwodu RC modelującego procesor składający się z N elementów (rdzeni, jednostek obliczeniowych, bloków funkcjonalnych). Każdy węzeł takiego obwodu odpowiada elementowi procesora, obudowy, radiatora lub materiału termoprzewodzącego. T_{amb} oznacza temperaturę otoczenia[1] (ang. *ambient*). Najczęściej zakłada się, że temperatura otoczenia jest stała lub zmienia się dużo wolniej niż temperatura samego procesora. W razie potrzeby, można zamodelować temperaturę wewnątrz obudowy komputera (temperaturę otoczenia) jako kolejny węzeł sieci przypisując mu rezystancję termiczną i pojemność cieplną [97]. T_{C1} do T_{CN} oznacza temperaturę poszczególnych rdzeni. R oznacza rezystancję termiczną, a C pojemność cieplną. Należy zaznaczyć, że z uwagi na ciepło właściwe i wymiary, pojemność cieplna obudowy wraz z radiatorem C_P jest 2 do 3 rzędów większa niż pojemność cieplna samego procesora.

Skupiony model termiczny opisać można równaniem stanu [34, 57]:

$$\frac{d\mathbf{T}}{dt} = A\mathbf{T} + B\mathbf{P}(s, \mathbf{T}, t), \tag{3.1}$$

gdzie $\mathbf{T}$ i $\mathbf{P}$ są wektorami kolumnowymi o wymiarze N_{RC}, a A oraz B są kwadratowymi macierzami stopnia N_{RC}. Wektor stanu $\mathbf{T}$ zawiera wartość temperatury w każdym

[1]W niniejszej pracy, dotyczącej temperatury procesora, temperatura otoczenia oznacza temperaturę powietrza w obudowie, w bezpośrednim otoczeniu procesora, a nie temperaturę powietrza w pomieszczeniu, w którym ten komputer pracuje.

węźle termicznej sieci RC. Wektor $\mathbf{P}(s, T, t)$ zawiera wartości mocy rozpraszanej przez każdy węzeł w chwili t. $\mathbf{P}$ zawiera wartości niezerowe jedynie dla węzłów sieci odpowiadających elementom samego procesora, jako że obudowa i radiator są elementami pasywnymi. Moc chwilowa rozpraszana przez procesor zależy od 3 parametrów: aktywności procesora, czyli liczby operacji które wykonuje, szybkości s, czyli pary (f, V), częstotliwości pracy i przypisanego jej napięcia zasilania, oraz temperatury. Symulatory termiczne działające w oparciu o równanie stanu (3.1) wykorzystują zwykle iteracyjne metody rozwiązywania równań różniczkowych zwyczajnych do obliczania temperatury procesora w kolejnych dyskretnych momentach.

Model termiczny komputera testowego

Użyty w niniejszej pracy komputer testowy zawiera czterordzeniowy procesor Core 2 Quad Q9400 firmy Intel. Zawiera on 6 MB pamięci podręcznej drugiego poziomu na łącznej powierzchni 164 mm^2. Powierzchnia jednego rdzenia wynosi około 20 mm^2. Przy maksymalnej mocy rozpraszanej przez jeden rdzeń na poziomie $20W$, średnia gęstość mocy rdzeni wynosi ok 1 W/mm^2. Cały procesor składa się z dwóch jednakowych modułów w jednej obudowie, z których każdy zawiera po dwa rdzenie i duży blok pamięci podręcznych. Każdemu z rdzeni przyporządkowano jeden z węzłów termicznej sieci RC. Podobnie, dwa węzły modelu odpowiadają dwóm częściom pamięci podręcznej poziomu drugiego umieszczonym w odrębnych modułach. Kolejne węzły przyporządkowano obudowie, warstwie materiału termoprzewodzącego i radiatorowi.

Wymiary obudowy procesora odczytano z jego dokumentacji [98]. W parametrach modelu termicznego, uwzględniony został materiał, z którego wykonano radiator (aluminium). Przewodność cieplna krzemu jest zależna od temperatury [7], ale w zakresie temperatur charakterystycznych dla komputerów klasy PC utrzymuje się w granicach od 150 do 120 W(m·K). Przyjęto średnią wartość 135 W/(m·K) dla uproszczenia modelu. Całkowitą rezystancję termiczną R_{tot} od procesora do otoczenia obliczono na podstawie różnicy temperatur procesora przy minimalnym i maksymalnym obciążeniu. Pomiar mocy rozpraszanej przez procesor opisany jest w rozdziale 3.3.

Tabela 3.1: Parametry procesora testowego

parametr	wartość
liczba rdzeni	4
częstotliwość zegara	2.66; 2.33; 2.00 GHz
pamięć L2	6 MB
zakres napięć	0.8500 – 1.3625 V
czas zmiany napięcia	10 μs
powierzchnia procesora	164 mm^2
grubość krzemu	0.5 mm
szerokość obudowy	30 mm
szerokość radiatora	60 mm
temperatura maksymalna	74°C
temperatura krytyczna	100°C
TDP	95 W

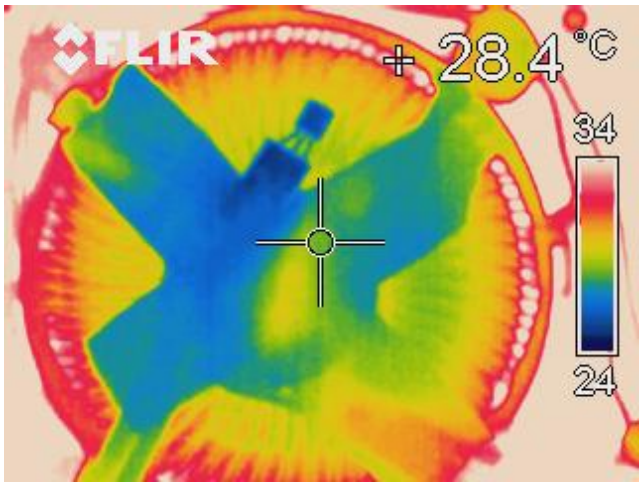
3.2 Dynamiczne zarządzanie temperaturą w komputerach

Efektywna implementacja mechanizmu sterującego temperaturą procesora wymaga wykorzystania czujników temperatury umieszczonych bezpośrednio w strukturze procesora. Ponieważ różnice temperatury na powierzchni procesora mogą przekraczać kilkanaście Kelwinów [92], wnioskowanie o temperaturze całego procesora na podstawie tylko jednego czujnika będzie prowadziło do utraty informacji. Co za tym idzie nie jest wykluczone przegrzewanie się procesora jeśli nie będzie zapewniony odpowiedni „zapas termiczny". Oznacza to, że przy zadanym ograniczeniu temperatury maksymalnej procesor pracował będzie z wydajnością o wiele mniejszą niż jest to teoretycznie możliwe.

Implementując mechanizm dynamicznego zarządzania temperaturą w systemie operacyjnym można założyć, że czujniki temperatury umieszczone są w najgorętszych punktach poszczególnych rdzeni procesora, ewentualnie są wyskalowane tak, aby podawać temperaturę w tych punktach. Czujniki temperatury umieszczone bezpośrednio w rdzeniu procesora mają dwie zasadnicze wady. Po pierwsze, zajmują cenną powierzchnię procesora, ponieważ należy umieszczać je w miejscach o największej

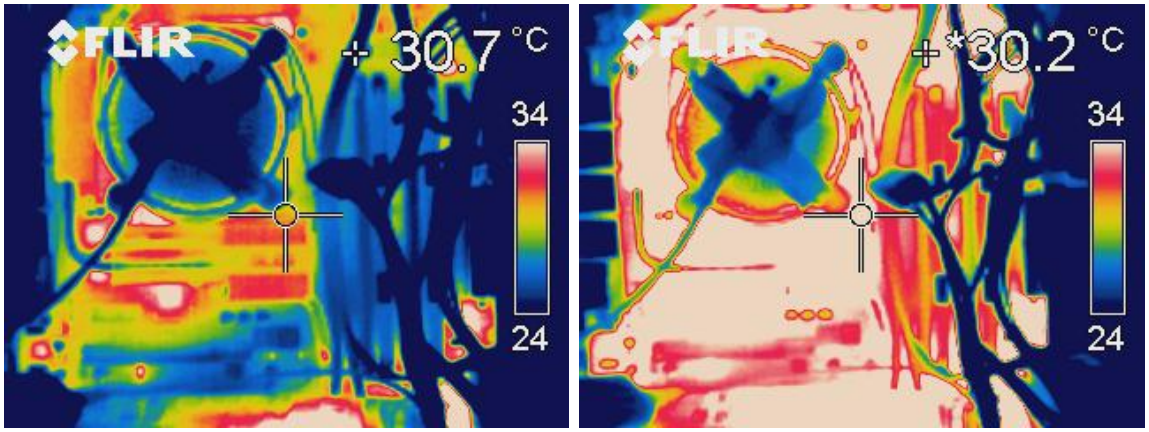
Tabela 3.2: Parametry modelu termicznego

parametr	wartość
przewodność cieplna krzemu	135 W/(m·K)
ciepło właściwe krzemu	$1.75 \cdot 10^6$ J/(m^3·K)
przewodność cieplna radiatora (aluminium)	237 W/(m·K)
ciepło właściwe radiatora	$2.42 \cdot 10^6$ J/(m^3·K)
rezystancja cieplna R_{tot}	0.45 K/W
pojemność cieplna C_r	88.88 J/K



Rysunek 3.5: Umiejscowienie czujnika temperatury w strumieniu powietrza wlotowego do radiatora

aktywności, a więc i temperaturze [99]. Z drugiej strony, w tych miejscach powierzchnia układu jest najbardziej cenna; znajdują się tam ścieżki krytyczne i umieszczenie czujnika może negatywnie wpływać na wydajność procesora [100]. Po drugie, czujniki temperatury same rozpraszają energię, przyczyniając się do wzrostu temperatury w krytycznych miejscach. Ponieważ dostępność czujników temperatury jest ograniczona, można zastosować model termiczny i energetyczny procesora i na podstawie sprzętowych liczników zdarzeń standardowo umieszczonych w procesorze próbować aproksymować temperaturę w wybranych punktach [101, 102]. Zasadniczą wadą takiego podejścia jest znaczny narzut obliczeń poświęconych na aktualizację modelu termicznego procesora.



a) b)

Rysunek 3.6: Wnętrze komputera bezpośrednio po włączeniu (a) i po 5 minutach od uruchomienia testu obciążeniowego – czterech instancji programu `burnP6` (b)

3.2.1 Sprzętowe mechanizmy zabezpieczające w procesorach ogólnego przeznaczenia

W procesorach firmy Intel, mającej największy udział na rynku procesorów przeznaczonych do stacji roboczych, serwerów i zastosowań wysokiej wydajności (ang. *High Performance Computing – HPC*), zaimplementowano szereg sprzętowych zabezpieczeń przed termicznym uszkodzeniem procesora [103]. Do historycznie najstarszego mechanizmu, który w odpowiedzi na przegrzanie wyłącza procesor, dodano 2 nowsze nazwane `TM1` i `TM2` (ang. *Thermal Monitor 1, 2*). Aktywowane są w niższej temperaturze, to jest zanim wystąpi awaryjne wyłączenie procesora, i automatycznie obniżają jego temperaturę kosztem utraty wydajności.

Kiedy temperatura przekracza zdefiniowany poziom powyżej którego procesorowi grozi uszkodzenie sygnał `THERMTRIP#` jest aktywowany, wstrzymywane są wszystkie linie zegarowe i odłączane jest napięcie zasilania V_{CC} rdzenia procesora. Kolejne 2 mechanizmy, oznaczane jako `TM1` oraz `TM2` oferują coraz większy poziom wyrafinowania. Automatyczny, sprzętowy mechanizm kontroli temperatury `TM1` wstrzymuje sygnał zegarowy w kolejnych odcinkach czasu z odpowiednio dobranym wypełnieniem, w ten sposób ograniczając temperaturę procesora. Wadą tego mechanizmu jest znaczna utrata wydajności przetwarzania oraz zwiększony czas obsługi przerwań. Przerwania przychodzące w czasie, gdy zegar jest wyłączony, są obsługiwane dopiero

po przywróceniu jego działania.

Mechanizm TM2 działa podobnie do poprzedniego, z tą różnicą, że do ograniczenia energii rozpraszanej w jednostce czasu wykorzystuje skalowanie napięcia i częstotliwości (ang. *Dynamic Voltage and Frequency Scaling – DVFS*). Po przekroczeniu zadanego progu temperatury jeden z powyższych mechanizmów pozostaje aktywowany do czasu osiągnięcia poziomu temperatury poniżej zadanego progu i nie krócej niż przez czas w przybliżeniu równy 1 ms [103].

3.2.2 Odczyt temperatury

W nowych procesorach wielordzeniowych standardem jest umożliwienie odczytania temperatury każdego z rdzeni. Jest to możliwe dzięki umieszczeniu co najmniej po jednym czujniku temperatury na rdzeń. Poszczególne odczyty dokonywane są z rozdzielczością 1 K, a ich dokładność zależy od aktualnej temperatury [104]. Z powodu rozrzutu parametrów w procesie produkcyjnym, dokładność czujników temperatury jest zmienna i może być różna w poszczególnych egzemplarzach procesorów. Uśrednienie wyników n kolejnych pomiarów pozwala na zmniejszenie szumu, zwiększenie precyzji oraz dokładności.

Istnieje wiele metod odczytania wyników pomiaru temperatury z czujników dostępnych w procesorze. W systemach operacyjnych z jądrem Linux w wersji 2.6 oraz 3.X dostępny jest moduł o nazwie coretemp. Przekazuje on informacje o odczytanej temperaturze bezpośrednio do systemu plików. Aktualną oraz krytyczną temperaturę każdego rdzenia można odczytać z plików dostępnych w: /sys/devices/platform/coretemp.N, gdzie N jest indeksem rdzenia, począwszy od 0.

W architekturze IA-32 istnieją dwie instrukcje do odczytu i zapisu rejestrów specjalnych (MSR). Są to RDMSR oraz WRMSR. RDMSR odczytuje wskazany rejestr specjalny, a wynikowe 64 bity umieszcza w rejestrach EAX:EDX. Analogicznie instrukcja WRMSR zapisuje wskazany rejestr zawartością rejestrów EAX:EDX. Rejestrem kontrolującym działanie sprzętowego układu kontroli temperatury jest IA32_THERM_INTERRUPT. Maksymalną dopuszczalną temperaturę dla danego procesora można odczytać z rejestru IA32_TEMPERATURE_TARGET. Aktualna temperatura danego rdzenia procesora zapisana jest na bitach 22:16 w rejestrze IA32_THERM_STATUS [103].

3.2.3 Sterowanie napięciem zasilania i częstotliwością pracy procesora

Nowoczesne procesory mogą pracować na kilku (zwykle od 2 do około 10) dostępnych poziomach wydajności. Każdemu z nich odpowiada ustalony poziom napięcia zasilania i częstotliwości pracy rdzenia procesora. Umożliwia to ograniczenie zużycia energii w czasie niskiego obciążenia zadaniami, dzięki zależności mocy dynamicznej od kwadratu napięcia zasilania. Zwykle też istnieje możliwość niezależnego ustawienia poziomu napięcia zasilania każdego rdzenia procesora z osobna. W architekturze IA-32 wybór ustawienia realizowany jest przez zapis do rejestru `IA32_PERF_CTL`. W jądrze systemu `Linux`, dostępny jest wygodny interfejs w postaci modułu `cpufreq`. Zmiana częstotliwości pracy polega na wpisaniu do pliku `/sys/devices/system/cpu/cpuN/cpufreq/scaling_setspeed` (gdzie `N` jest indeksem rdzenia) częstotliwości w kHz wybranej spośród dostępnych. Czas zmiany ustawienia napięcia i zasilania wynosi zazwyczaj około 10 μs.[2] W tym czasie dany rdzeń nie przetwarza instrukcji. Wpływ zmiany częstotliwości pracy na wydajność zostanie przeanalizowany w dalszej części pracy.

3.2.4 Migracja wątków

W zależności od obciążenia systemu zadaniami, migracja wątków może być bardziej skutecznym narzędziem dynamicznego zarządzania temperaturą procesora niż DVFS. Jest tak zwłaszcza w przypadkach, gdy liczba aktywnych zadań jest znacząco niższa niż liczba rdzeni procesora. W takich przypadkach występują duże gradienty temperatury na powierzchni procesora. Własność tę można wykorzystać przenosząc wątki z gorących rdzeni na chłodniejsze – mniej obciążone, bądź aktualnie nieużywane. Są dwa powody strat wydajności podczas przenoszenia zadania na inny rdzeń. Pierwszy odpowiada czasowi skopiowania kontekstu zadania na docelowy rdzeń. Drugi stanowi tak zwany efekt *zimnych pamięci* – przenoszony na nowy rdzeń jest jedynie kontekst procesu, bez zawartości pamięci podręcznych, układu prognozy rozgałęzień oraz buforów TLB. Dlatego obserwuje się czasową stratę wydajności wynikającą z dodatkowych chybień w pamięciach podręcznych. Sumaryczny narzut związany z mi-

[2] W systemie `Linux` 2.6 oraz 3.X opóźnienie (w nanosekundach) związane ze zmianą szybkości pracy procesora dostępne jest w pliku `/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_transition_latency`.

gracją zadania wynosi zwykle pomiędzy $2\,\mu s$, a $1\,ms$, w zależności od zadania [19]. Znaczącą wadą migracji wątków jest zmienność tego narzutu pomiędzy programami. Ważne jest również, że dodatkowe chybienia w pamięciach podręcznych mają negatywny wpływ na wydajność całego systemu, z powodu blokowania magistrali pamięci głównej. Jeśli narzut związany z przeniesieniem wątku i efektem *zimnych pamięci* jest niższy niż związany z wykonywaniem programu przy niższej niż domyślna (maksymalna) częstotliwości, migracja wątków jest preferowana.

Żeby przydzielić wybrany wątek do rdzenia lub podzbioru rdzeni procesora, można w systemach z jądrem `Linux` zastosować program `taskset`. Za jego pomocą możliwe jest przypisanie aktualnie wykonywanego zadania do wybranego rdzenia procesora lub zbioru rdzeni. Odczytując cyklicznie temperaturę poszczególnych rdzeni i przenosząc zadania z bardziej gorących na chłodniejsze rdzenie, można uzyskać proste narzędzie do migracji wątków. Główną zaletą takiego rozwiązania jest prostota jego implementacji. Polepszenie jego skuteczności można uzyskać integrując algorytm migracji wątków z mechanizmem odpowiedzialnym za szeregowanie zadań. Dzięki temu można wykorzystać informacje o temperaturze bezpośrednio w procesie szeregowania zadań i ograniczyć liczbę niepotrzebnych migracji.

3.3 Charakterystyka termiczna komputera

Zbudowanie skutecznego mechanizmu dynamicznego zarządzania temperaturą i jego ocena wymagają określenia dokładnego modelu termicznego komputera, a w szczególności procesora. Utworzenie termicznego modelu zachowania procesora pozwala na jego ekstrapolację na przyszłe generacje procesorów o podobnej architekturze, ale odmiennych parametrach, np. większej liczbie rdzeni. Ponadto, efektywny mechanizm sterowania pracą procesora przy ograniczeniu temperaturowym musi opierać się na prognozie temperatury, jak wspomniano w rozdziale 2.3.

Model termiczny procesora opisany w rozdziale 3.1 składa się z rezystancji i pojemności termicznych odpowiadających elementom procesora wraz z obudową oraz radiatorem połączonych w termiczny obwód RC. Oprócz elementów pasywnych występują w nim źródła ciepła – aktywne elementy procesora podczas wykonywania operacji zużywają energię i rozpraszają ją w postaci ciepła. Całkowita moc rozpraszana przez procesor zawiera tak zwaną dynamiczną oraz statyczną składową i może

być opisana wzorem [105]:

$$P = aCV^2 f + \tau aV I_{short} f + V I_{leak}. \tag{3.2}$$

W układach CMOS dynamiczna moc rozpraszana P_{dyn} jest sumą mocy rozpraszanej wskutek oddziaływania 2 czynników. Pierwszy wynika z ładowania i rozładowywania pojemności elektrycznych C obciążających wyjście każdej bramki. Składowa druga odpowiada prądowi zwarcia płynącemu przez bardzo krótki czas τ podczas przełączania się bramki. Moc statyczna P_{stat} nie zależy bezpośrednio od tego, czy procesor wykonuje w danym momencie operacje i stanowi funkcję temperatury i napięcia zasilania [106]. Składowa statyczna związana jest z prądem upływu I_{leak} (częściowym przewodzeniu prądu przez tranzystor przy napięciu niższym niż napięcie progowe V_{TH}) oraz tunelowaniem nośników przez izolator bramki [6]. Sumarycznie moc rozpraszaną przez każdy blok funkcjonalny procesora opisać można następującym równaniem:

$$P(a, V, f, T) = P_{dyn}(a, V, f) + P_s(V, T) \tag{3.3}$$

Jednym z parametrów, który ma dominujący wpływ na wielkość mocy dynamicznej jest współczynnik aktywności a. W ogólności oznacza on średni procent bramek logicznych, która zmienia swój stan w każdym cyklu. W nowoczesnych procesorach tylko niewielka część tranzystorów może być przełączana w każdym cyklu. Jest to spowodowane ograniczoną równoległością na poziomie instrukcji (ang. *Instruction-Level Parallelism – ILP*), ograniczonym zrównolegleniem zadań oraz ograniczeniami wynikającymi z mocy i temperatury [107]. Tak zdefiniowany współczynnik aktywności jest niewygodny w modelowaniu na poziomie mikroarchitektury procesora. Na najwyższym poziomie abstrakcji procesor możemy podzielić na 3 następujące typy obszarów: układy wykonawcze (rdzenie procesora), pamięci oraz pozostałe elementy, odpowiedzialne między innymi za komunikację z pamięcią. W przypadku każdego z tych typów należy ustalić współczynnik na poziomie architektury komputera odpowiadający aktywności. W przypadku rdzeni procesora bardzo dużą korelację z mocą wykazuje współczynnik określający średnią liczbę instrukcji zakończoną podczas jednego cyklu zegara – IPC [108]. Podobnie, w przypadku pamięci ilość rozpraszanej energii jest proporcjonalna do liczby odwołań do niej.

Można argumentować, że uzależnianie zużycia energii przez rdzeń procesora tylko od jednego współczynnika (IPC) może prowadzić do niedokładności. Po pierwsze, w przypadku różnych programów dynamiczny rozkład wykonanych instrukcji może

być bardzo różny. Wykonanie instrukcji zmiennoprzecinkowej wymaga zaangażowania rozbudowanego układu zmiennoprzecinkowego, a np. wykonanie instrukcji logicznych jest relatywnie szybsze i powoduje zużycie mniejszej ilości energii. Duża liczba instrukcji wymagających pobrania danych z pamięci podręcznych będzie powodowała zużycie energii w tych pamięciach, tak więc moc rozpraszana przez pamięci podręczne będzie pewną funkcją rozmiaru zbioru roboczego programu. Kolejnym aspektem jest zużycie energii na instrukcje, które zostaną anulowane z powodu błędnej prognozy rozgałęzienia programu. W zależności od skuteczności układu prognozy rozgałęzień, długości potoku procesora oraz częstotliwości występowania skoków warunkowych rośnie ilość energii poświęconej na wykonywanie instrukcji, których wyniki nie zostaną wykorzystane.

W praktyce [109] najczęściej dominują proste instrukcje (pobranie danych z pamięci, porównania, skoki warunkowe, dodawanie), a różnica energii przypadająca na jedną instrukcję dla różnych programów nie jest znaczna. Między innymi z uwagi na efektywność energetyczną stosuje się w nowoczesnych układach krótsze potoki wykonawcze, co skutkuje mniejszą wrażliwością na błędy prognozy rozgałęzień. Potoki o długości sięgającej 30 etapów (więcej niż pięciokrotność średniej długości bloku podstawowego (ang. *basic block*) [109]) należą do przeszłości.

Ponadto, wraz ze zmniejszaniem się wymiaru bazowego technologii fabrykacji układów VLSI, zmniejszają się wymiary całych rdzeni procesora. W technologii 45 nm rozmiar poszczególnych rdzeni w procesorach segmentu konsumenckiego[3] nie przekracza 30 mm^2 [107]. Dodatkowo, biorąc pod uwagę efekt przestrzennego filtrowania dolnoprzepustowego temperatury (opisany w rozdziale 2), można traktować rdzeń procesora jako pojedynczy element opisany średnią aktywnością i mocą. W efekcie, dla zadanej mikroarchitektury procesora, współczynnik IPC stanowi rozsądną miarę aktywności pojedynczego rdzenia.

3.3.1 Pomiar mocy

Moc rozpraszaną przez komputer mierzono miernikiem sieciowym `LUMEL N10A`. Pozwala on na odczyt mocy co 1 s. Podstawowym źródłem niedokładności jest wpływ całego systemu komputerowego. Oprócz procesora, prąd zużywany jest na zasilanie

[3] W odróżnieniu od specjalizowanych procesorów serwerowych, np. z rodziny `Itanium` firmy `Intel`, albo rodziny `Power` firmy `IBM`.

płyty głównej, karty graficznej, pamięci głównej (RAM) oraz dysków. Ponadto sam zasilacz ma ograniczoną sprawność, która jest nieliniową funkcją obciążenia. Nawet pomiar prądu bezpośrednio na 12 V linii zasilającej procesor obarczony jest błędem z powodu skończonej sprawności konwerterów napięć [110]. Jak zostanie pokazane, mimo dużego błędu pomiaru, uzyskany model zużycia energii przez procesor jest wystarczająco dokładny dla potrzeb symulacji termicznej i prognozy temperatury w procesorach wielordzeniowych.

Do wykonania wszystkich pomiarów mocy, temperatury i wydajności wykorzystano autorski program w języku C. Wykonuje on odczyt rejestrów MSR procesora w celu ustalenia aktualnej liczby instrukcji wykonanych przez każdy z rdzeni, liczby chybień w pamięci podręcznej ostatniego poziomu oraz temperatury. Odczyt mocy odbywa się co 1 s z uwagi na ograniczenia miernika. Pozostałe wartości można odczytywać znacznie częściej. W praktyce, przy niewielkiej zajętości procesora (do 10% czasu jednego rdzenia), możliwe były odczyty nawet co około 100 μs. Nierówności odstępów czasu pomiędzy poszczególnymi odczytami, wynikające między innymi z obsługi przerwań uwzględniane są dzięki odczytom liczby cykli zegara procesora. Testowy procesor nie zawiera licznika cykli niezależnego od taktowania poszczególnych rdzeni procesora, ale uwzględniając informacje o taktowaniu poszczególnych rdzeni, można przeskalować odczytywane wartości.

W eksperymentach używane były 3 zestawy programów testowych: `SPEC CPU 2000` [111], `PARSEC` [112] oraz `cpuburn`. Programy z pakietu `cpuburn` wykonują nieskończoną pętlę napisaną tak, aby maksymalnie obciążyć procesor. W każdym cyklu wykonywana jest jednakowa liczba instrukcji. Zestaw `SPEC CPU 2000` składa się z 26 programów naukowych i inżynierskich podzielonych na dwie grupy: programy *stałoprzecinkowe* i *zmiennoprzecinkowe*[4] Wszystkie programy nastawione są na testowanie wydajności pamięci i procesora podczas wykonywania zadań jednowątkowych. Pomiędzy sobą poszczególne programy różnią się rozmiarem zbioru roboczego, wzorcami odwołań do pamięci a w konsekwencji liczbą jednocześnie przetwarzanych instrukcji. Dzięki temu można wśród nich wyróżnić programy *gorące*, które wymuszają wysoką aktywność procesora, duże zużycie energii i wysoką temperaturę, oraz programy *zimne*.

Zestaw `PARSEC` zawiera zróżnicowane aplikacje wielowątkowe. Nie ograniczają się one jedynie do zastosowań w obliczeniach wysokiej wydajności (ang. *High Performan-*

[4]Programy zmiennoprzecinkowe w odróżnieniu od stałoprzecinkowych zawierają znaczący udział rozkazów zmiennoprzecinkowych [109].

ce Computing). `PARSEC` jest reprezentatywny dla współczesnych programów z wielu dziedzin. Używanie w eksperymentach programów wielowątkowych jest pożądane z uwagi na upowszechnianie się procesorów wielordzeniowych i programów wielowątkowych oraz efekty związane z konieczną synchronizacją wątków i komunikacją między nimi.

Wszystkie testowane programy obciążają głównie CPU i pamięci RAM. Ponieważ komputer testowy nie wyświetla grafiki[5], można przyjąć jednostajne obciążenie urządzeń peryferyjnych (karta sieciowa, dysk twardy, karta graficzna) i stałe zużycie przez nie energii. Dzięki temu możliwy jest pośredni pomiar mocy rozpraszanej przez procesor. Całkowitą moc zużywaną przez komputer P_k można wyrazić jako sumę mocy pobieranej w stanie bezczynności P_{idle}, mocy dynamicznej procesora P_{dyn} zależnej od jego aktywności, mocy statycznej P_{stat} oraz mocy zużywanej przez pamięć operacyjną P_{mem}:

$$P_k = P_{idle} + P_{dyn} + P_{stat} + P_{mem}. \tag{3.4}$$

Odejmując od aktualnego odczytu mocy wartość zmierzoną w stanie bezczynności i utrzymując dwie z trzech pozostałych składowych na stałym poziomie, można obliczyć wartość trzeciej. Średnia moc rozpraszana przez komputer w stanie spoczynku wyniosła 83 W a temperatura procesora 41 °C.

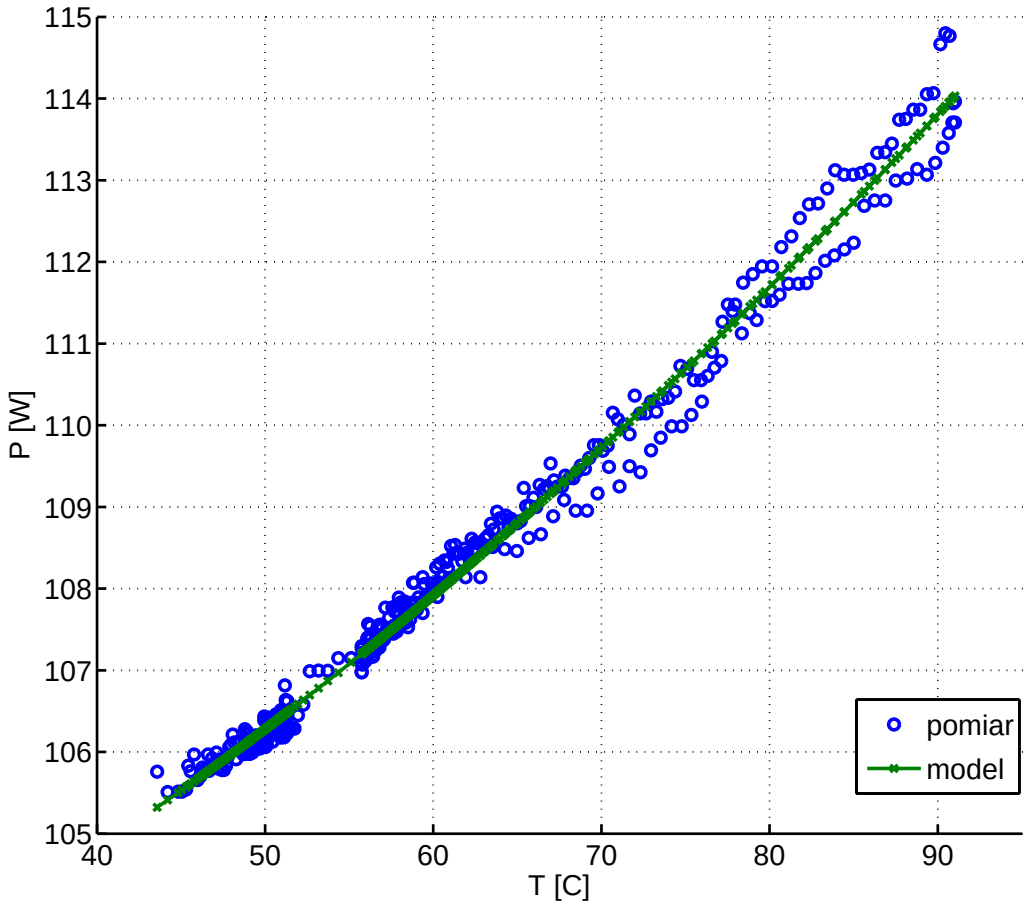
Moc procesora związana z prądem upływu zależy wykładniczo od temperatury. Zależność prądu upływu i temperatury może być wyrażona jako [106]:

$$I_{leak}(T) = I_{leak}(T_0) \cdot T^2 \cdot e^{(\alpha/T)}. \tag{3.5}$$

gdzie α jest parametrem technologicznym. W zakresach temperatur spotykanych w cywilnych zastosowaniach, zależność mocy statycznej rozpraszanej przez jeden z rdzeni daje się z dużą dokładnością aproksymować funkcją liniową. Pomiary zależności mocy statycznej od temperatury procesora przeprowadzono przy ustalonych obrotach wiatraka procesora i przy nieobciążonym procesorze. Przy pomocy strumienia gorącego powietrza skierowanego bezpośrednio na radiator, zmieniano warunki otoczenia i tym samym temperaturę procesora. Na rysunku 3.7 można zobaczyć wyniki pomiarów mocy zużywanej przez komputer w funkcji temperatury. Regresja zależności z równania (3.5) daje zależność, którą można w praktycznym zakresie temperatur (35–85°C) z bardzo małym błędem przybliżyć równaniem liniowym:

$$P_{stat}(T) = \frac{1}{N_c} \cdot \frac{P_k(85) - P_k(35)}{50} \cdot (T - 35) \tag{3.6}$$

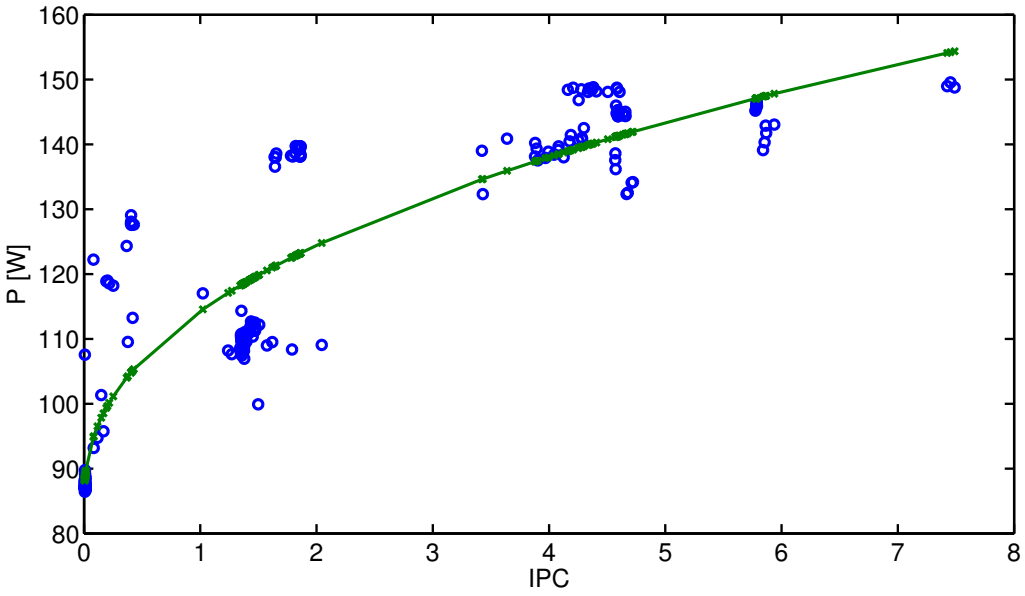
[5]Komunikacja z komputerem testowym odbywała się jedynie przez sieć Ethernet.



Rysunek 3.7: Zależny od temperatury składnik mocy rozpraszanej przez procesor

Moc rozpraszana przez rdzeń procesora składa się z części stałej oraz części zależnej w sposób nieliniowy od chwilowej aktywności mierzonej współczynnikiem IPC. Pewna część energii potrzebna jest na zasilanie poszczególnych elementów procesora (jednostka prognozy rozgałęzień, kolejka rozkazów, stacja rezerwacyjna, bufor kompletacji (ang. *reorder buffer*)) niezależnie od ich aktywności. Wraz ze wzrostem zajętości zużycie energii przypadające na jedną instrukcję spada. Stąd, można zamodelować moc rozpraszaną przez procesor jako zależną wykładniczo od IPC z wykładnikiem mniejszym od 1. Dodatkowo w niniejszej pracy uwzględniono ilość energii zużywaną przez pamięci, jako liniowo zależną od liczby chybień w pamięciach podręcznych ostatniego poziomu (LLCM). W poniższym wzorze N_c oznacza liczbę rdzeni procesora, a k_A do k_D są parametrami.

$$P_k = P_{idle} + k_A + k_B \cdot \sum_{c=1}^{N_c} IPC_c^{k_C} + k_D \cdot \sum_{c=1}^{N_c} LLCM_c \tag{3.7}$$



Rysunek 3.8: Moc rozpraszana przez komputer jako funkcja aktywności procesora

Aby znaleźć zależność mocy rozpraszanej przez poszczególne rdzenie procesora od ich aktywności mierzonej współczynnikiem IPC zmierzono wydajność oraz zużycie energii przez komputer podczas wykonywania całego zestawu testów `PARSEC`. Następnie zsumowano liczby wykonanych instrukcji oraz chybień w pamięciach podręcznych (LLCM) w każdym ze rdzeni. Programy wchodzące w skład zestawu `PARSEC` składają się z części równoległych jak i sekwencyjnych. Po uśrednieniu danych co 1 ms, zarejestrowano wartości współczynnika IPC w granicach od 0 do 8.

W celu dopasowania współczynników równania (3.7) do zebranych danych, została zastosowana regresja. Podzbiór danych, wybrany przy użyciu próbkowania równomiernego, oraz wynik dopasowania parametrów funkcji można zobaczyć na rys. 3.8. Moc dynamiczna rozpraszana przez pojedynczy rdzeń procesora testowego wyrażona może być jako

$$P_{core} = P_{core}^{idle} + \frac{1}{N_c} \cdot k_A + \frac{1}{N_c} \cdot k_B \cdot N_c^{k_C} \cdot IPC^{k_C} \tag{3.8}$$

Sumarycznie więc moc dynamiczna rozpraszana przez jeden rdzeń testowanego procesora wynosi w przybliżeniu $P_{core} = 7 + 10.71 \cdot IPC^{0.55}$ W.

Kolejnym ważnym elementem modelu jest założenie stałości temperatury otoczenia procesora ($T_{amb} = const$). W praktycznych zastosowaniach mamy do czynienia z wymuszonym obiegiem temperatury w obudowie komputera. Z tego powodu temperatura wewnątrz obudowy zmienia się w niewielkim stopniu. Ponadto tempo tej

zmiany jest znacznie mniejsze niż tempo zmian temperatury radiatora, więc przyjęcie stałej temperatury otoczenia (wewnątrz obudowy) stanowi akceptowalne uproszczenie. Natomiast w przypadku konkretnego systemu rozszerzenie modelu termicznego o pojemność cieplną obudowy komputera i jej rezystancję termiczną do otoczenia jest stosunkowo proste [97]. Pomimo założenia, że temperatura otoczenia zmienia się znacznie wolniej niż temperatura układu procesor-obudowa, konieczny jest pomiar temperatury otoczenia T_{amb}, czyli temperatury w obudowie komputera. Jest to spowodowane tym, że czujniki temperatury w procesorze podają wartości w skali bezwzględnej[6], a model termiczny operuje na temperaturach względem T_{amb}.

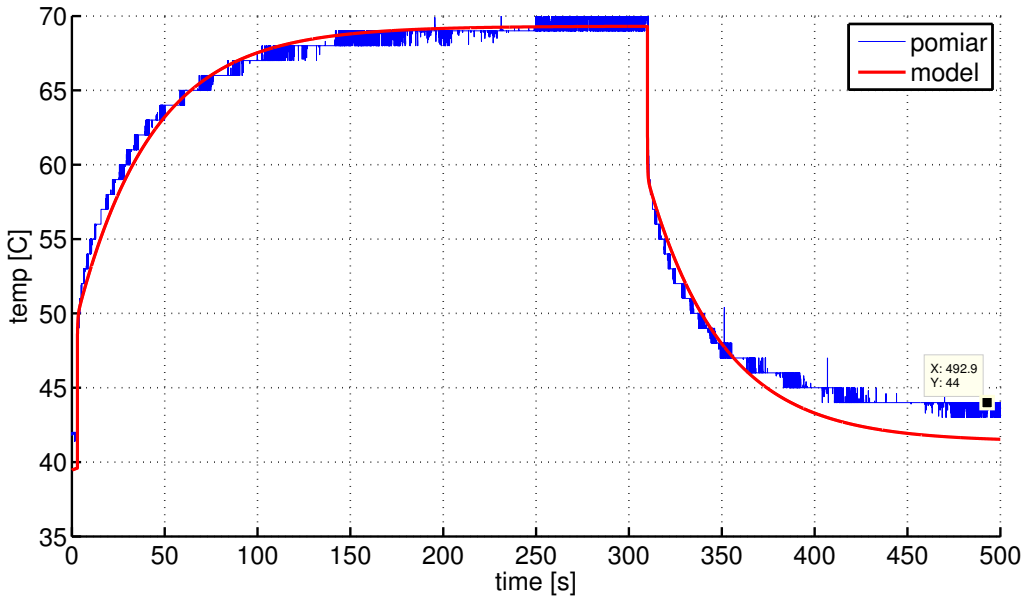
3.4 Weryfikacja modelu termicznego

W poniższym rozdziale przedstawione są wyniki weryfikacji modelu termicznego procesora przez porównanie wyników pomiarów temperatury procesora z wynikami symulacji przeprowadzonej w programie HotSpot w wersji 5.0 [20]. Parametry modelu termicznego zostały określone w rozdziale 3.1, a oszacowanie mocy rozpraszanej przez procesor w rozdziale 3.3.

W celu otrzymania pobudzenia o stałej wartości wykorzystano program `burnMMX` z pakietu `cpuburn`. Programy z tego pakietu wykonują w nieskończoność krótką pętlę zaprojektowaną tak, aby wykonywana była jednocześnie jak największa liczba instrukcji. Zaletą tego programu jest zapewnienie stałej aktywności procesora w czasie. Zapisano profil temperatury oraz wartości współczynnika IPC podczas wykonywania 4 instancji programu. Poszczególne wątki zostały uruchomione jednocześnie. Następnie, gdy temperatura procesora ustabilizowała się na maksymalnym poziomie, zostały wyłączone. Z profilu zapisanego na komputerze testowym odczytano wartość IPC. Na podstawie równania (3.8) obliczono moc rozpraszaną przez każdy z rdzeni w stanie bezczynności oraz podczas wykonywania programu `burnMMX`. Zapewnione zostały stałe obroty wentylatora, czyli stała rezystancja termiczna od procesora do otoczenia R_{tot} i temperatura otoczenia.

Na rysunku 3.9 przedstawione są wyniki porównania symulacji termicznej z pomiarami wykonanymi na komputerze testowym. Można zauważyć, że odpowiedź modelu

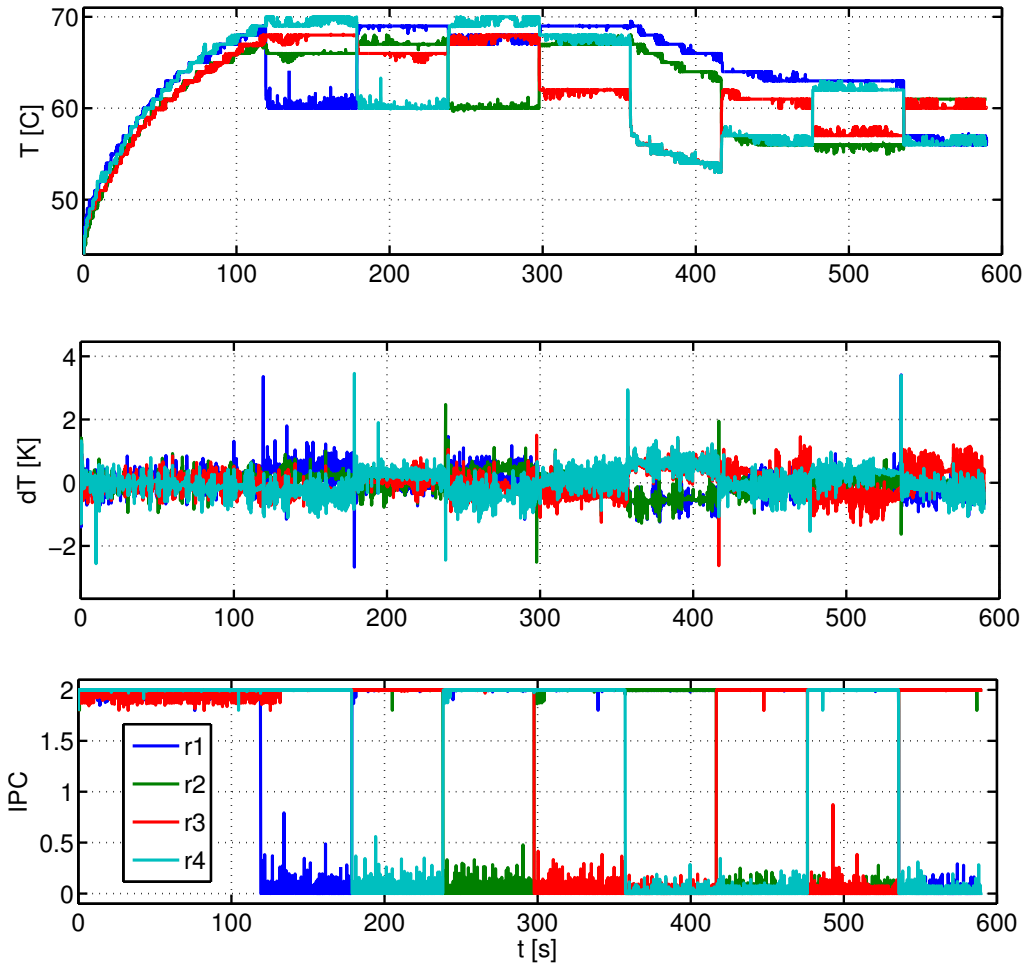
[6]Dokładniej, w postaci różnicy względem maksymalnej dopuszczalnej temperatury.



Rysunek 3.9: Porównanie modelu z wynikami pomiarów. Odpowiedź termiczna procesora na stałe pobudzenie.

na zadane pobudzenie dobrze odpowiada danym pomiarowym. Temperatura w stanie ustalonym jest równa zmierzonej. Widoczna różnica pomiędzy danymi pomiarowymi, a wynikiem symulacji w programie `HotSpot`,w końcowej części wykresu wynika ze zmiany temperatury otoczenia procesora. Zmiana ta nie została uwzględniona w symulatorze, ale wartość temperatury otoczenia jest jednym z parametrów samego modelu. Dobrze widoczne są również dwie wyraźne fazy wzrostu temperatury. Najpierw temperatura rośnie bardzo szybko co wynika z relatywnie niedużej pojemności cieplnej samego procesora. Następnie tempo wzrostu spada z powodu dużej stałej czasowej RC obudowy i radiatora.

Dokładniejszą weryfikację rezystancji termicznej w modelu przeprowadzono na podstawie profilu temperatury i aktywności procesora zebranego podczas wykonywania programu `burnP6` na wszystkich rdzeniach procesora, kolejnych trójkach oraz parach rdzeni. Pozwoliło to zweryfikować wartości rezystancji termicznej pomiędzy poszczególnymi składowymi procesora. Przykładowo, testowy procesor składa się z dwóch układów scalonych w jednej obudowie, tak więc rezystancja termiczna pomiędzy rdzeniami nr 2 i 3 jest znacznie większa niż pomiędzy rdzeniami leżącymi w jednym układzie scalonym. Przekłada się to na znacznie większą różnicę temperatur w procesorze w sytuacji gdy aktywna jest jedynie para rdzeni znajdująca się w jednym układzie



Rysunek 3.10: Porównanie modelu z wynikami pomiarów: temperatura rdzeni procesora, błąd modelu względem rzeczywistej temperatury i aktywność procesora dla zadanej sekwencji programów testowych.

scalonym.

Na rysunku 3.10 przedstawiono przebieg temperatury poszczególnych rdzeni, różnicę pomiędzy modelowaną temperaturą a rzeczywistą oraz aktywność poszczególnych rdzeni. Widoczne znaczące błędy podczas zmiany aktywnych rdzeni wynikają z przyjętego długiego czasu aktualizacji danych wejściowych modelu wynoszącego 100 ms. W przedziale od 360 do 420 s obciążone były rdzenie 1 i 2. Według modelu rdzenie aktywne powinny mieć większą niż w rzeczywistości temperaturę, podczas gdy rdzenie pasywne mniejszą. Błąd ten wynika z przyjętego uproszczenia modelu polegającego na przypisaniu całej obudowie procesora jednego węzła w sieci RC. Decyzja ta podyktowana została niską rezystancją termiczną w metalowej obudowie i potrze-

bą ograniczenia złożoności modelu. W przypadku bardzo dużej rezystancji termicznej w procesorze rezystancja termiczna w obudowie nie jest zaniedbywalna i w zaprezentowanej sytuacji może być źródłem błędu.

Poprawność zastosowania modelu skupionego (ang. *lumped model*) można zgrubnie ocenić obliczając liczbę Biota Bi jako [7]:

$$Bi = \frac{hL_c}{k} \tag{3.9}$$

gdzie $L_c = V/A$ jest długością charakterystyczną, V to objętość obiektu, A jest polem powierzchni wymiany ciepła, h jest współczynnikiem przenikania ciepła, a k to przewodność cieplna. W praktyce liczba Biota (Bi) oznacza stosunek rezystancji cieplnej wewnątrz rozpatrywanego elementu do rezystancji na jego powierzchni. Dla Bi=0 analiza dyskretna jest ścisła. Zakłada się, że analiza z użyciem modelu skupionego daje dobre wyniki przy $Bi \leqslant 0.1$ [7].

Ponieważ najważniejsza jest dokładność obliczenia temperatury samego procesora potrzebna jest analiza dokładności modelu dla jego elementów składowych. Obliczenie Bi dla poszczególnych bloków funkcjonalnych procesora wymaga określenia ich wymiarów fizycznych, przewodności cieplnej k krzemu oraz współczynnika przenikalności cieplnej h. Wymiary poszczególnych bloków można odczytać lub oszacować z dokumentacji procesora (na przykład grubość krzemu) oraz zdjęć przedstawiających topologię procesora. Problematyczne może być określenie współczynnika przenikalności cieplnej h:

$$h = \frac{Q}{A \cdot \Delta T} \tag{3.10}$$

gdzie Q jest ciepłem, a ΔT różnicą temperatur pomiędzy obiektem a jego otoczeniem. Aby oszacować h należało określić różnicę temperatur pomiędzy powierzchnią procesora a jego obudową podczas wykonywania ustalonego programu przez procesor. Ponieważ wykonanie bezpośredniego pomiaru wymaga ingerencji w strukturę procesora, posłużono się symulacją w programie `HotSpot`. Przyjęto również Q równe mocy rozpraszanej przez jeden rdzeń procesora podczas wykonywania programu testowego. Oszacowany w ten sposób współczynnik przenikalności cieplnej wynosi $9.1 \cdot 10^4 \frac{W}{m^2 \cdot K}$. Ostatecznie dla parametrów podanych w tabelach 3.1 oraz 3.2 oszacowana liczba Biota wynosi 0.45. Wynika stąd ograniczona dokładność symulacji.[7] Ponieważ wymagana

[7]Można zwiększyć dokładność dzieląc w modelu dodatkowo krzem procesora na warstwy. Należy zaznaczyć, że znacząca poprawa dokładności będzie miała miejsce jedynie przy bardzo dużych gęstościach mocy.

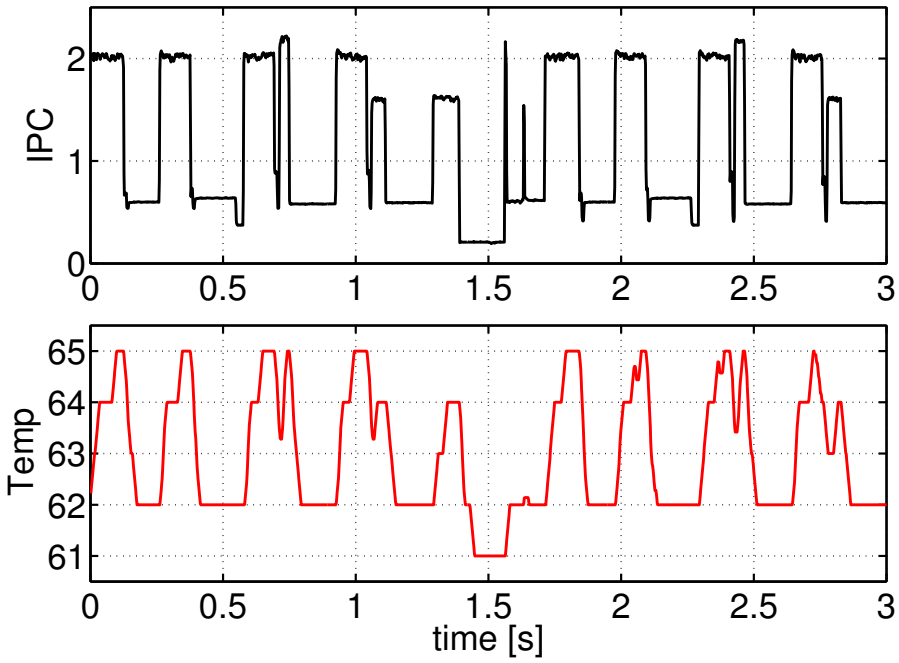
dokładność prognozy temperatury jest niewielka, bardziej istotne jest zapewnienie szybkości prognozy. W celu zwiększenia dokładności można podzielić rdzeń procesora na osobne węzły odpowiadające jego poszczególnym składowym na poziomie mikroarchitektury. Kolejną możliwością jest równomierne dzielenie bloków na części [113]. Prowadzi to jednak do wzrostu złożoności modelu i powoduje wydłużenie czasu obliczeń.

Skupiony model termiczny wykorzystujący sieć RC może być z powodzeniem wykorzystany do symulacji termicznej procesorów [34, 87, 96]. Przyjmując dostatecznie mały okres czasu t_s można na podstawie wektorów temperatury i mocy w chwili t_i oraz równania 3.1 obliczyć temperaturę $\mathbf{T}(t_{i+1})$. Powyższy model termiczny procesora zostanie wykorzystany do prognozy temperatury w rozdziale 4. Tam też zostanie przedstawiona analiza wpływu długości podstawowego kroku symulacji na dokładność obliczonej prognozy temperatury.

3.5 Modelowanie dynamicznego zachowania programów

Dynamiczne zachowanie programu można opisać średnią liczbą zdarzeń odczytanych z poszczególnych sprzętowych liczników (PMC) oraz mocą zużywaną przez rdzeń procesora. Poszczególne programy różnią się między sobą znacząco aktywnością, wykorzystaniem pamięci a także rozpraszaną mocą podczas ich wykonywania, co przekłada się na różne zachowanie termiczne. Przetwarzane programy przechodzą przez szereg *faz*, w których zachowanie wątku i rdzenia znacząco się różnią [114–116]. Zmiany zachowania programu, a więc i poszczególne fazy odpowiadają przetwarzanym fragmentom kodu: funkcjom i procedurom wywoływanym cyklicznie. Iteracyjne wykonywanie skutkuje bardzo często cyklicznym zachowaniem programów. Dzieje się tak najczęściej w przypadku programów zmiennoprzecinkowych, takich jak aplikacje naukowe i inżynierskie, w których pewien algorytm wykonywany jest iteracyjnie.

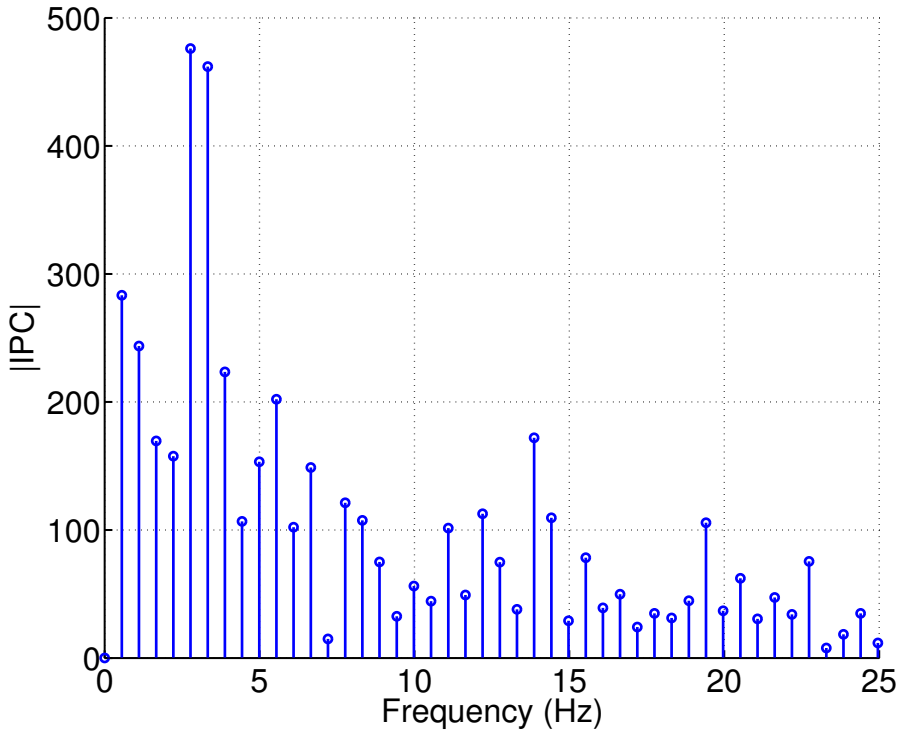
Wykonywane programy mają hierarchiczną strukturę: składają się z funkcji i procedur, które wywołują inne funkcje. Taka struktura jest oparta na blokach podstawowych (ang. *basic blocks*) – fragmentach kodu między kolejnymi instrukcjami skoków. Z tego powodu zachowanie programów zmienia się w różnych skalach czasowych od sekund począwszy aż do ułamków mikrosekund. Różnice w wartościach współczyn-



Rysunek 3.11: IPC i przebieg temperatury zmierzony podczas 3 sekund wykonywania programu `apsi` na komputerze testowym

nika IPC w poszczególnych fazach wykonywanych programów mają 2 przyczyny. Po pierwsze, różne fragmenty kodu mogą mieć różny stopień równoległości poziomu instrukcji (ILP) wynikający na przykład z występowania konfliktów dostępu do zasobów procesora (strukturalnych) lub konfliktów sterowania. Drugim czynnikiem jest oczekiwanie na przesłanie danych z i do pamięci. Chybienie w pamięci podręcznej wiąże się z koniecznością pobrania danych z pamięci głównej. Wpływ dużej liczby takich zdarzeń na wartość IPC można zaobserwować monitorując sprzętowe liczniki zdarzeń. Zmiany zachowania programów w zakresie wykorzystania pamięci podręcznych mogą być wykorzystywane, przez dostosowanie rozmiaru tych pamięci do bieżących potrzeb programów, w celu ograniczenia zużycia energii przez procesor [117].

Zmiany aktywności wątku wykonywanego przez rdzeń przekładają się bezpośrednio na chwilowe zużycie energii. W konsekwencji obserwowane są wyraźne zmiany temperatury rdzenia procesora podczas wykonywania jednego programu. Przykładowo, z wykresów na rys. 3.11 można odczytać fragment przebiegu IPC oraz temperatury procesora w czasie wykonywania programu `apsi`. Dane te zostały zarejestrowane na komputerze testowym, po ustaleniu się temperatury radiatora. IPC wątku zmienia się w granicach od 0,2 do 2,1, co przekłada się na różnicę w mocy rozpraszanej przez



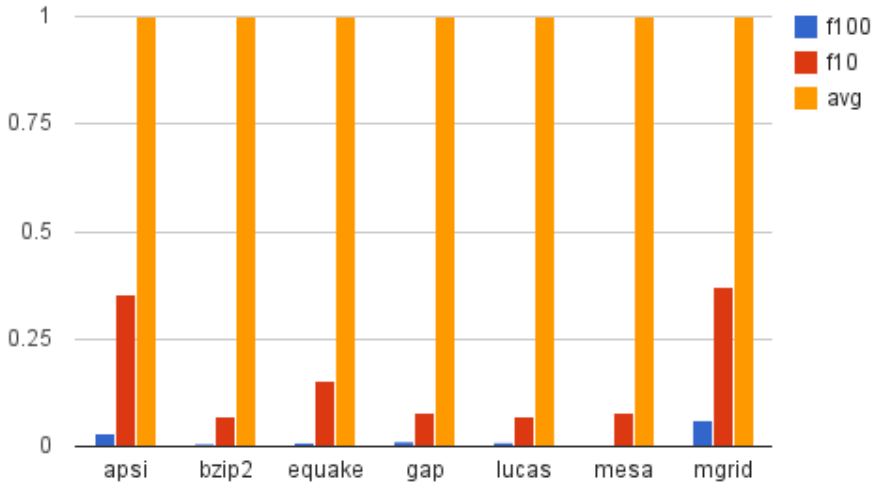
Rysunek 3.12: Profil częstotliwości IPC programu `apsi`

dany rdzeń procesora, pomiędzy minimalnym a maksymalnym poziomem aktywności wynoszącą około stu procent. Można zobaczyć, że zmiany IPC wątku przekładają się bezpośrednio na zmiany temperatury. Ponieważ stała czasowa RC rdzenia procesora jest na poziomie milisekund, to krótkotrwałe zmiany w wykorzystaniu energii są dokładnie odwzorowane przez temperaturę. Zmiany długotrwałe, na przykład uruchomienie nowego zadania i jego zakończenie, wpływają na zmianę temperatury całego układu składającego się z procesora, obudowy i radiatora.

Ponieważ procesor działa jak termiczny filtr RC, zmiany aktywności, których czas trwania jest znacząco krótszy niż stała czasowa rdzenia nie przekładają się na temperaturę procesora. Częstotliwość graniczna takiego filtru może być obliczona jako:

$$f_c = \frac{1}{2\pi RC} \tag{3.11}$$

Przyjmując stałą czasową RC rdzenia równą 1 ms otrzymujemy częstotliwość graniczną f_c wynoszącą 159 Hz. Częstotliwość ta odpowiada okresowi potrzebnemu na wykonanie przeciętnie kilku do kilkudziesięciu milionów instrukcji. Oznacza to, że znacząco krótsze fragmenty programu nie mają wyraźnego wpływu na temperaturę procesora.



Rysunek 3.13: Znormalizowany błąd średniokwadratowy aproksymacji przebiegu IPC wartością średnią (*avg*) oraz rozwinięciem Fouriera z 10 oraz 100 współczynnikami

Najprostszym sposobem na modelowanie zachowania poszczególnych programów dla celów symulacji termicznej jest przypisanie każdemu z nich skalarnej wartości określającej uśrednione zużycie mocy lub średnią aktywność. Niestety przyjęcie modelu średniego IPC może być podczas symulacji termicznej źródłem błędów [118]. W fazach o wyższym niż średnie zużyciu energii, temperatura może przekroczyć maksymalne dopuszczalne wartości. Podobnie nieuwzględnienie faz o niższym niż średnie zużyciu energii spowoduje pominięcie możliwości wykorzystania powstającego zapasu termicznego do zwiększenia wydajności procesora.

W symulatorze termicznym można wykorzystać zapisany profil aktywności procesu uzyskany z pomiarów na komputerze lub z dokładnej symulacji czasowej (ang. *cycle-accurate*). Zapis przebiegu temperatury jest jednak niewygodny w użyciu. Wymaga przechowywania dużej ilości danych a przekształcenia na nim (np. zmiana częstotliwości zegara symulowanego procesora) wymagają zmiany całego profilu. Profil częstotliwości IPC przykładowego programu został przedstawiony na rysunku 3.12. Można na nim dostrzec dominujące częstotliwości zobrazowane na przebiegu programu (rys. 3.11). Cykliczne zachowanie wielu programów pozwala zastosować przybliżenie ich przebiegu szeregami Fouriera [119]. Wzór (3.12) określa wartość szeregu w chwili t. Współczynniki $\{a_0, .., a_K\}$ oraz $\{b_1, .., b_K\}$ wraz z okresem T wystarczają do opisania

aktywności wątku.

$$s_K(t) = a_0 + \sum_{k=1}^{K} a_k \cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{K} b_k \sin\left(\frac{2\pi kt}{T}\right) \tag{3.12}$$

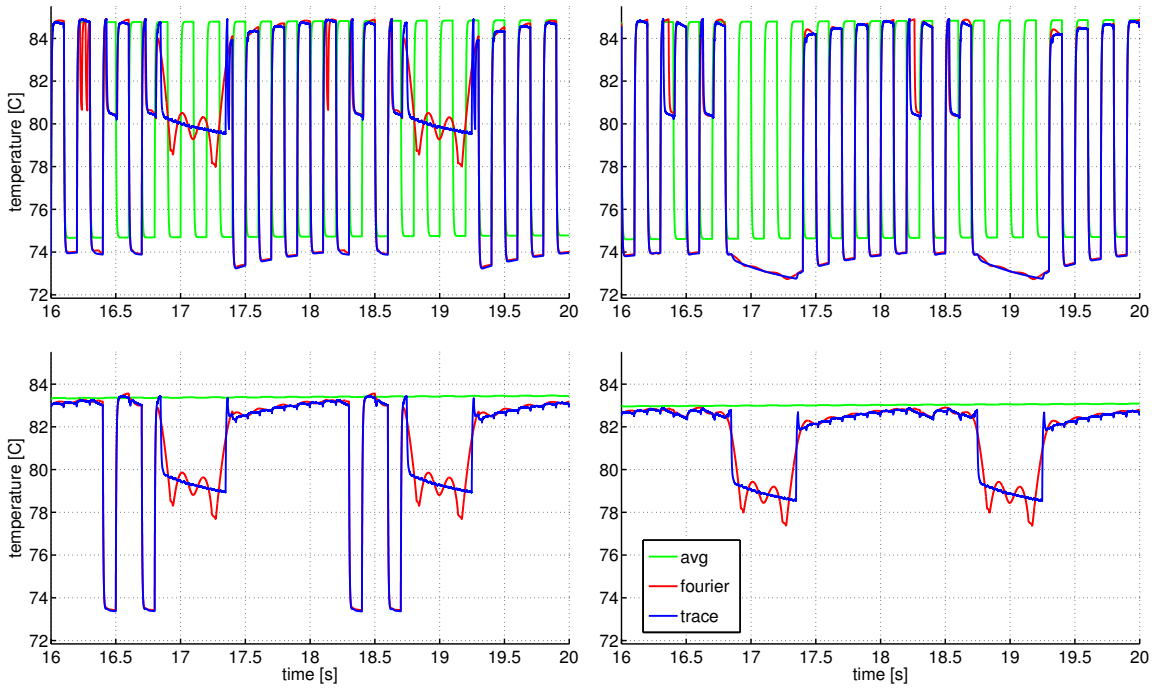
Głównymi zaletami wykorzystania współczynników szeregu Fouriera do opisu zachowania wątków są łatwość wykonywania przekształceń oraz możliwość bardzo szybkiego odczytania informacji o właściwościach wątku: wartości średniej oraz zmienności jego zachowania w czasie. Dodatkowo, mała zajętość pamięci, duża dokładność odwzorowania i niski koszt obliczenia zachowania wątku w dowolnym momencie przemawiają za wykorzystaniem takiego wysokopoziomowego modelu wątku.

Ocenę jakości przybliżenia profilu aktywności wątku rozwinięciem szeregu Fouriera wykonano w oparciu o wartość błędu średniokwadratowego względem rzeczywistego profilu dla przybliżenia wartością średnią, oraz szeregami używając 10 oraz 100 współczynników. Na rysunku 3.13 przedstawiono znormalizowany błąd średniokwadratowy przybliżenia zachowania siedmiu różnych programów z zestawu `SPEC CPU 2000`. Można zauważyć znaczną przewagę przybliżenia przebiegu programów szeregami Fouriera, nawet przy niewielkiej liczbie współczynników.

Aby ocenić wpływ modelu zachowania wątku na dokładność symulacji termicznych przeprowadzono eksperyment w którym porównano wyniki symulacji termicznej systemu z czterordzeniowym procesorem wykorzystując trzy różne modele zadań. Pierwszy z nich to model wykorzystujący średnią wartość aktywności zadania, w drugim model przybliżający aktywność zadania rozwinięciem szeregu Fouriera, w trzecim – przebieg zapisany z komputera.

W eksperymencie wykorzystano symulator termiczny `MAGMA` [55, 87]. Symulator ten jest oparty na modelu `HotSpot` i umożliwia symulację termiczną systemów wielordzeniowych wykorzystujących mechanizmy dynamicznego zarządzania temperaturą, w tym skalowanie częstotliwości i migrację wątków. `MAGMA` nie pełni roli symulatora funkcjonalnego, a jedynie modeluje zachowanie termiczne procesora na podstawie danych o mocy rozpraszanej przez jego poszczególne bloki funkcjonalne.

Na rysunku 3.14 przestawiono fragment wyników symulacji termicznej procesora. Poszczególne wykresy przedstawiają najwyższą temperaturę każdego z rdzeni podczas symulacji z trzema różnymi modelami zadań: średnią *avg*, zapisany przebieg *trace* i oparty na szeregach Fouriera. Można zauważyć że wykorzystanie wartości średniej skutkuje innym zachowaniem termicznym systemu niż dwa pozostałe modele. Między



Rysunek 3.14: Fragment przebiegu temperatury w czterech rdzeniach procesora uzyskany podczas symulacji wykorzystującej różne modele zachowania zadania

innymi nie pozwala na przewidzenie okresów, w których temperatura poszczególnych rdzeni obniża się.

Z powodu dużej zmienności zachowania programów w czasie oraz dużego zróżnicowania charakterystyk pomiędzy nimi, ocena wydajności nie może być oparta jedynie na częstotliwości pracy procesora. Przykładowo, w pracy [44] autorzy oceniają skuteczność metody DTM na podstawie sumarycznej częstotliwości poszczególnych rdzeni. Wykorzystanie częstotliwości rdzenia jako kryterium wydajności, prowadzi do zwiększania częstotliwości rdzeni przetwarzających programy o niskim współczynniku IPC, ponieważ moc rozpraszana przez nie jest niższa niż moc rdzeni przetwarzających dużo instrukcji. W efekcie sumaryczna częstotliwość rdzeni procesora wzrośnie, przy jednoczesnym spadku przepustowości całego procesora. Co więcej, w przypadku zadań wielowątkowych nawet ocena wydajności uwzględniająca IPC procesu może być myląca [90].

Analizy zawarte w powyższym podrozdziale zaczerpnięto z pracy [119].

3.6 Modelowanie dynamicznego skalowania napięcia i częstotliwości

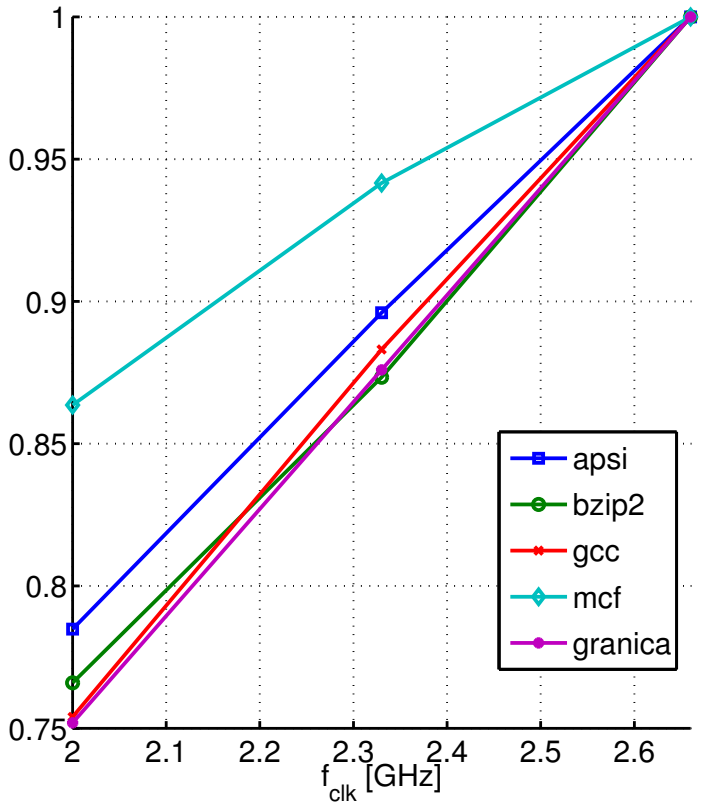
Dynamiczne skalowanie napięcia i częstotliwości (DVFS) jest stosowane w celu oszczędzania energii oraz ograniczenia temperatury procesora. Wpływ skalowania zegara na czas przetwarzania różni się pomiędzy poszczególnymi programami. Zarówno efektywne wykorzystanie skalowania częstotliwości jak i dokładne modelowanie temperatury procesorów wymaga informacji o wpływie częstotliwości pracy rdzenia na wydajność przetwarzania oraz zużyciu energii.

3.6.1 Wpływ DVFS na wydajność

Średnia liczba instrukcji zakończonych w cyklu pracy rdzenia procesora nie jest liniową funkcją częstotliwości jego zegara. Podczas pracy procesora programy przechodzą przez fazy, w czasie których głównym ograniczeniem wydajności jest procesor oraz takie, w czasie których współczynnik IPC gwałtownie spada z powodu dużej liczby odwołań do pamięci [120]. Przy niezerowej liczbie odwołań do pamięci głównej, zmniejszenie częstotliwości zegara powoduje wzrost IPC. Ponieważ średni czas odwołania do pamięci się nie zmienia, maleje liczba cykli procesora poświęcana na oczekiwanie na dane z pamięci. Im większa liczba chybień w pamięci podręcznej ostatniego poziomu (odwołań do pamięci głównej), tym wzrost IPC jest większy przy takim samym obniżeniu częstotliwości zegara. Równoważnie, jeśli mierzyć wydajność przetwarzania danego zadania, im większa liczba chybień w pamięciach podręcznych tym negatywny wpływ zmniejszenia częstotliwości zegara jest mniejszy.

W celu opracowania modelu wiążącego wydajność rdzenia procesora z częstotliwością jego pracy posłużono się danymi zebranymi podczas działania zestawu testów `PARSEC` i `SPEC CPU 2000`, tak jak to zostało opisane w rozdziale 3.3. Opracowanie modelu wiążącego chwilową wartość IPC zadania z częstotliwością procesora pozwala na zwiększenie dokładności modelu mocy rdzenia $P(T, IPC(f), f, V)$. Wzrost IPC wraz ze zmniejszeniem częstotliwości f jest funkcją liczby chybień w pamięci podręcznej najwyższego poziomu (LLCM).

Na rysunku 3.15 przedstawiono wyniki pomiarów znormalizowanej szybkości przetwarzania (odwrotności czasu wykonania) wybranych programów z pakietu `SPEC CPU`. Można zauważyć, że spadek wydajności związany ze zmianą częstotliwości jest zawsze



Rysunek 3.15: Zmniejszenie szybkości przetwarzania programów przy zmianie częstotliwości pracy, znormalizowane względem maksymalnej częstotliwości

niższy niż wynikałoby to jedynie ze spowolnienia pracy samego rdzenia. Ponadto, średnią szybkość przetwarzania można z dużą dokładnością modelować funkcją liniową. W przypadku programu `mcf`, który charakteryzuje się bardzo dużą liczbą odwołań do pamięci głównej (wysoki współczynniki LLCM), spowolnienie jest znacznie mniejsze niż ma to miejsce w pozostałych przypadkach.

W tabeli 3.3 zebrano czasy działania oraz średnie współczynniki IPC i LLCM zmierzone podczas wykonywania wybranych programów przy różnych częstotliwościach zegara. Widoczny jest duży rozrzut aktywności procesora i liczby odwołań do pamięci głównej pomiędzy poszczególnymi programami.

Czas przetwarzania programu składa się, w przybliżeniu, z czasu poświęconego na wykonanie instrukcji oraz czasu oczekiwania na dane z pamięci głównej. Na tej podstawie sformułowano zależność łączącą szybkość przetwarzania danego programu

Tabela 3.3: Średnie wartości współczynników IPC i LLCM oraz czasy wykonania wybranych programów z pakietów SPEC CPU 2000 oraz PARSEC przy różnych ustawieniach taktowania procesora

				czas [s]		
program		IPC($f_{\max}$)	LLCM($f_{\max}$)	2,00 GHz	2,33 GHz	2,66 GHz
SPEC	apsi	1.10	$2.69 \cdot 10^{-3}$	151,5	132,7	118,9
	bzip2	1.42	$8.25 \cdot 10^{-4}$	101,7	89,2	77,9
	gcc	1.26	$4.71 \cdot 10^{-4}$	56,1	47,9	42,3
	mcf	0.18	$8.05 \cdot 10^{-3}$	115,8	106,2	100,0
PARSEC	bodytrack	1.37	$4.25 \cdot 10^{-5}$	4,9	4,2	3,7
	canneal	0.259	$3.14 \cdot 10^{-3}$	11,4	10,6	9,9
	dedup	1.40	$3.46 \cdot 10^{-4}$	13,3	11,4	10,1
	facesim	1.39	$7.22 \cdot 10^{-4}$	9,9	8,6	7,7

z częstotliwością oraz liczbą chybień w pamięciach podręcznych ostatniego poziomu

$$s(f) = s(f_{\max}) \cdot \frac{f}{f_{\max}} + \left(1 - \frac{f}{f_{\max}}\right) LLCM \cdot k_E. \tag{3.13}$$

Na podstawie danych zebranych podczas działania programów z pakietów SPEC CPU 2000 oraz PARSEC wyliczono współczynnik k_E. Ponieważ szybkość przetwarzania odpowiada liczbie instrukcji wykonanych w ustalonym czasie, stąd zależność określającą IPC w funkcji częstotliwości rdzenia procesora określić można następująco:

$$IPC(f) = IPC(f_{\max}) + k_E \cdot LLCM \cdot \left(\frac{f_{\max}}{f} - 1\right). \tag{3.14}$$

Wzór (3.14) można stosować w szybkich symulacjach termicznych procesorów wykorzystujących mechanizmy dynamicznego skalowania napięcia i częstotliwości do kontrolowania temperatury.

3.6.2 Wpływ DVFS na moc

Dynamiczna moc rozpraszana przez procesor zależy liniowo od częstotliwości zegara oraz od kwadratu napięcia zasilania. Ten fakt, oraz opisany w poprzednim podrozdziale mniejszy spadek wydajności niż spadek taktowania zegara sprawiają, że DVFS jest bardzo skutecznym mechanizmem ograniczającym zużycie energii.

Ponieważ poszczególne poziomy napięć w danym procesorze jednoznacznie przyporządkowane są do częstotliwości, wykorzystywanie obu zmiennych nie jest konieczne. Ponadto różne napięcia odpowiadające ustalonym częstotliwościom mogą się różnić pomiędzy poszczególnymi egzemplarzami [98]. Przyjęto następujący model mocy dynamicznej rdzenia przy zastosowaniu DVFS:

$$P_{dyn} = P_{dyn}(f_{max}) \cdot \left(\frac{f}{f_{max}} \right)^{k_B} . \tag{3.15}$$

W celu określenia zależności zużywanej energii od wybranej częstotliwości wykonano pomiary mocy rozpraszanej przez procesor podczas wykonywania programu `burnMMX`, który charakteryzuje się stałym w czasie i niezależnym od częstotliwości zegara współczynnikiem IPC. Następnie dopasowano parametry modelu do danych uzyskanych z pomiarów. W przypadku komputera wybranego do testów, uzyskany współczynnik k_B wynosi 2.45.

3.7 Wnioski

Z modelu `HotSpot` wynika niższa temperatura rdzeni znajdujących się na brzegach lub rogach układu scalonego. Tym czasem w testowanym procesorze (schematyczna topografia układu na rysunku (3.4)), przy jednakowym obciążeniu wszystkich rdzeni, te umieszczone w rogach chipu mają wyższą temperaturę. Częściowo różnice te można tłumaczyć niedokładnością czujników temperatury. Ponadto odczytywana wysokość temperatury może być również uzależniona od miejsca położenia czujnika w rdzeniu.

Model termiczny procesora opisuje temperaturę względem temperatury otoczenia. W krótkim okresie czasu można uznać ją za stałą, co pozwala uprościć obliczenia. Jednakże, nawet w dobrze wentylowanej obudowie komputera, zaobserwowano znaczne zmiany temperatury spowodowane długotrwałymi różnicami w poziomie mocy rozpraszanej przez procesor. Oznacza to, że w przypadku symulacji termicznych nie można zlekceważyć wpływu temperatury otoczenia. Należy również zaznaczyć, że tempo zmian temperatury otoczenia jest wielokrotnie mniejsze niż tempo zmian temperatury samego procesora. Procesy termiczne w komputerze naturalnie manifestują się w wielu skalach czasowych. Wynika stąd konieczność modelowania hierarchicznego, łączącego wydajność symulacji, elastyczność oraz dokładność.

Temperatura jest nieliniową funkcją mocy, a ta jest nieliniową funkcją wydajności, w dodatku zależną od aktualnie przetwarzanego zadania. Dlatego też szybka

i jednocześnie dokładna symulacja procesora wymaga uwzględnienia wydajności przetwarzanych zadań.

Rozdział 4

Prognoza temperatury procesorów wielordzeniowych

Prognozowanie temperatury poprawia skuteczność mechanizmów DTM przez ograniczenie czasu utrzymania zbyt wysokiej temperatury. Umożliwia to nie tylko zmniejszenie strat wydajności występujących w przypadku stosowania rektywnych mechanizmów lecz także umożliwia zapewnienie wysokiej wydajności chwilowej. Ponadto, prognozowanie temperatury procesora wielordzeniowego zwiększa efektywność zarządzania jego pracą, pozwalając na chwilowe zwiększenie wydajności w okresach, w których temperatura jest obniżana.

W razie braku prognozy temperatury procesora jej nieunikniony nagły wzrost spowoduje czasową pracę powyżej dopuszczonego limitu temperatury. Brak prognozy wymaga częstego odczytywania czujników temperatury w celu wykrycia sytuacji niepożądanych – przekroczenia wyznaczonej temperatury maksymalnej lub pracy poniżej możliwej wydajności. W odpowiedzi na przekroczenie temperatury maksymalnej konieczna jest aktywacja mechanizmu DTM wstrzymującego działanie procesora, bądź znaczne spowolnienie przetwarzania. Może to być nieakceptowalne, na przykład w systemach czasu rzeczywistego. Algorytm wykorzystujący prognozę dobiera ustawienia szybkości rdzeni procesora maksymalizując wydajność przy zadanym ograniczeniu temperatury maksymalnej.

4.1 Założenia

Omawiając mechanizm prognozy temperatury procesora nie można pominąć jego właściwości, takich jak dokładność i użyteczny czas prognozy, narzut związany z obliczeniami stosowanymi w modelu termicznym czy możliwość weryfikacji temperatury poprzez odczyt czujników umieszczonych w strukturze procesora oraz ograniczeń wynikających z parametrów współczesnych systemów komputerowych.

Rozważania dotyczące prognozy temperatury procesorów wielordzeniowych przedstawione w niniejszym rozdziale dotyczą procesorów wielordzeniowych o wysokiej wydajności, chłodzonych powietrzem i przeznaczonych do zastosowań w komputerach osobistych i stacjach roboczych. Wysoka wydajność przekłada się na dużą moc rozpraszaną przez procesor osiągającą łącznie od kilkudziesięciu do około 130 W.

Możliwe jest zbudowanie analogicznego modelu termicznego oraz mechanizmów prognozy dla procesorów przeznaczonych do zastosowań mobilnych. Jednakże z uwagi na drastycznie inne parametry układu chłodzenia (relatywnie mniejsza pojemność cieplna radiatora, wykorzystanie ciepłowodów (ang. *heat pipe*) oraz niższe moce) nie można automatycznie rozciągać przedstawionych tu wniosków na procesory przeznaczone do innych zastosowań. Nie bez znaczenia są również inne kryteria optymalizacji wynikające z ograniczonego zasobu energii oraz przewaga programów interaktywnych uruchamianych na urządzeniach przenośnych. Wymusza to inne strategie zarządzania pracą procesora.

4.1.1 Czujniki temperatury

Budowa

W masowo produkowanych procesorach wysokiej wydajności stosowane są dwa podstawowe typy czujników temperatury – cyfrowe i analogowe [100]. W czujnikach cyfrowych stosowane są oscylatory pierścieniowe. Pierścień złożony z nieparzystej liczby logicznych bramek **not** generuje oscylacje, których częstotliwość zależna jest od temperatury. Oscylacje te zliczane są w ustalonym czasie i na podstawie danych uzyskanych z uprzedniej kalibracji, przeliczane na temperaturę. Przykładem procesora, w którym zaimplementowano takie czujniki jest `IBM Power 5` [121].

Działanie analogowych czujników temperatury jest oparte na wykorzystaniu proporcjonalnej zależności właściwości elektrycznych półprzewodników od temperatury.

Mogą być być wykonane z użyciem diody termicznej, referencyjnego źródła prądu, komparatora prądowego i przetwornika analogowo-cyfrowego [100]. Przykładowo procesor `AMD Opteron` wyposażony jest w 38 takich czujników [122] rozmieszczonych w czterech rdzeniach oraz wspólnych blokach procesora. Wartość temperatury odczytywana jest na podstawie spadku napięcia przy wymuszonym prądzie.

Liczba czujników

Dwurdzeniowy procesor `POWER 5` zawiera 24 czujniki temperatury [121], a ośmiordzeniowy `POWER 7` aż 44 – w tym po pięć w każdym z rdzeni. Nowe procesory firmy `Intel` wyposażone są w 1 lub 2 czujniki w strukturze każdego rdzenia. Wspomniany wyżej procesor `AMD Opteron` odczytuje temperaturę w 8 punktach w każdym rdzeniu. Duża liczba czujników wynika z możliwych znacznych różnic temperatury na powierzchni procesora oraz różnego rozłożenia gorących punktów na jego powierzchni w zależności o przetwarzanego programu [92].

Ograniczenie liczby czujników temperatury w strukturze rdzenia procesora wynika przede wszystkim z ograniczenia powierzchni układu, którą można na nie przeznaczyć. Drugim czynnikiem jest rozpraszanie energii przez czujniki. Ponieważ powinny one znajdować się w miejscach o najwyższej temperaturze, oba te zasoby są szczególnie cenne.

Zakłada się, że czujniki implementowane we współczesnych procesorach cechuje dokładność rzędu $\pm 2^{\circ}$C przy rozdzielczości 1°C [123]. Problemem może być szum wpływający na odczyt z czujnika temperatury. Jednym ze sposobów jego kompensacji jest zwielokrotnienie pomiarów i uśrednienie odczytu. Wiąże się to jednak z opóźnieniem odczytu oraz wydatkowaniem dodatkowej energii na sam odczyt [124]. Stabilność i dokładność pomiaru temperatury można poprawić powiększając elementy, z których zbudowane są czujniki oraz zwiększając ich liczbę. Tak więc na wczesnym etapie projektowania nowego procesora konieczna jest symulacja termiczna procesora na poziomie mikroarchitektury w celu ustalenia gdzie potencjalnie mogą występować gorące punkty i gdzie należy umieścić czujniki temperatury.

Z powodu rozrzutu parametrów występującego w procesie technologicznym produkcji nowoczesnych układów CMOS, zaimplementowane czujniki wymagają kalibracji. Deklarowana przez producenta dokładność odczytu może być zależna od aktualnego poziomu temperatury – najczęściej dokładność rośnie w pobliżu $T_{\max}$ [104].

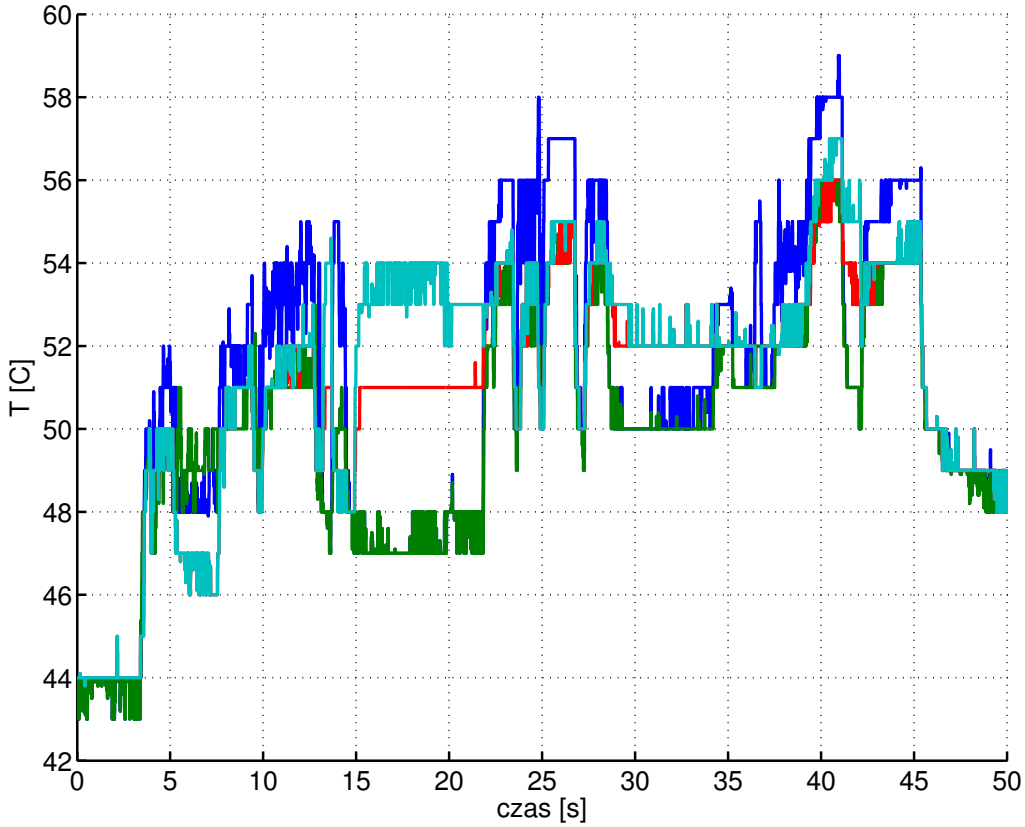
Sposobem na zwiększenie dokładności odczytu temperatury w razie niedostatecznej liczby czujników może być wykorzystanie modeli termicznych. Ponosząc znaczny koszt związany z aktualizowaniem modelu termicznego na bieżąco, możliwa jest aproksymacja temperatury w wybranych punktach na podstawie ograniczonej liczby czujników [102] lub szacowanie temperatury na podstawie liczników aktywności poszczególnych bloków funkcjonalnych procesora [101, 125].

4.1.2 Cel prognozy

Rzeczywistym celem prognozy temperatury procesora jest podjęcie właściwej decyzji przez mechanizm DTM. Oznacza to, że prognoza może być uważana za poprawną, jeśli używany algorytm zarządzający działaniem procesora zwrócił takie ustawienia, jakie zwróciłby w danym momencie mając dokładną informację o przyszłej temperaturze procesora. Wynika z tego, że odpowiednimi metrykami do oceny algorytmu prognozy są wyniki uzyskane pośrednio, na przykład maksymalna temperatura, czas wykonania testowego zestawu programów, czy sumaryczny czas kiedy temperatura przekraczała wartość maksymalną $T_{\max}$. Innym sposobem oceny algorytmu prognozy może być zliczanie błędnych prognoz. Za błędną możemy uznać taką, która skutkuje podjęciem złej decyzji, na przykład podwyższeniem częstotliwości rdzenia procesora, w następstwie którego przekroczona zostanie maksymalna dozwolona temperatura.

Ocena skuteczności prognozy oparta na wynikach uzyskanych pośrednio, na przykład na podstawie czasu przetwarzania zadania, może być upośledzona przez nieskuteczny algorytm sterowania pracą procesora pod ograniczeniem temperaturowym. Ponadto, ponieważ przebieg wykonania programu wielowątkowego na procesorze wielordzeniowym pod kontrolą wielozadaniowego systemu operacyjnego ma charakter stochastyczny, więc uzyskanie dokładnie takiego samego zachowania całego systemu, w celu porównania działania algorytmu, nie jest możliwe.

Szybka ocena i analiza skuteczności algorytmu prognozy możliwa jest na podstawie profili temperatury i obciążenia procesora zebranych podczas wykonywania programów testowych. Profile te zawierają informacje o temperaturze rdzeni procesora, liczbie chybień w pamięciach podręcznych ostatniego poziomu oraz liczbie wykonanych w każdym z rdzeni instrukcji odczytane co ustalony okres. Pomiary przeprowadzono podczas wykonywania zadań złożonych z wielowątkowych programów z zestawu `PARSEC` oraz zestawów od 1 do 4 programów z pakietu `SPEC CPU 2000`.



Rysunek 4.1: Profil temperatury czterech rdzeni procesora podczas wykonywania zestawu programów `PARSEC` z testowymi danymi wejściowymi

Przykładowy przebieg temperatury czterech rdzeni procesora przedstawiono na rysunku 4.1. Na jego podstawie można zauważyć gwałtowne zmiany temperatury całego procesora. W zależności od aktualnie wykonywanych zadań, występują duże różnice temperatury pomiędzy poszczególnymi rdzeniami, ponadto zadania o czasie przetwarzania znacząco krótszym niż stała czasowa RC obudowy procesora wraz z radiatorem ($\sim$100 s) nie powodują wzrostu temperatury do maksymalnego dostępnego poziomu.

Zawyżanie temperatury podczas prognozy może skutkować niepotrzebnym uruchomieniem mechanizmów DTM co przekłada się na utratę wydajności. Jeśli wykorzystanym mechanizmem jest skalowanie napięcia i częstotliwości (DVFS), zmniejsza się dodatkowo temperatura co powoduje jednocześnie wzrost poziomu niezawodności i ograniczenie prądu upływu. Natomiast częste i niepotrzebne migracje wątków skutkują zmniejszeniem efektywności przetwarzania i obniżeniem wydajności.

Tabela 4.1: Prognoza temperatury na podstawie ostatniej wartości: błąd średniokwadratowy (MSE) i maksymalny błąd prognozy ($|\varepsilon|_{\max}$) w °C dla różnych okresów

	czas prognozy [ms]					
	1	5	10	20	40	100
`SPEC CPU 2000 int`						
MSE	0.0018	0.0322	0.0905	0.1658	0.2371	0.3261
$\lvert\varepsilon\rvert_{\max}$	0.3	1.4	2.4	3.0	4.0	4.0
`SPEC CPU 2000 fp`						
MSE	0.0013	0.0234	0.0560	0.1252	0.2078	0.3787
$\lvert\varepsilon\rvert_{\max}$	0.4	1.4	2.6	3.8	4.9	5.2
`PARSEC -n 4`						
MSE	0.0013	0.0238	0.0678	0.1324	0.2086	0.2698
$\lvert\varepsilon\rvert_{\max}$	0.7	3.3	4.7	6.0	6.0	8.0

Zaniżona prognoza skutkować może brakiem wymaganej reakcji na przegrzewanie się procesora. Praca w wyższej niż przewidziano temperaturze skutkuje zmniejszeniem niezawodności. Wysoka temperatura wpływa na zwiększenie wielkości prądu upływu. Powoduje to zużywanie dodatkowej energii co z kolei wzmaga problemy z utrzymaniem temperatury na bezpiecznym poziomie. Dlatego też zaniżenie prognozy temperatury ma potencjalnie groźniejsze skutki niż jej zawyżenie.

Żeby można było skutecznie wykorzystać informację o prognozowanej temperaturze, musi być ona dostępna z odpowiednim wyprzedzeniem. Podstawowym kryterium wyboru okresu prognozy temperatury jest częstotliwość przerwania zegarowego – okres prognozy musi być dłuższy niż okres odpowiadający częstotliwości szeregowania zadań. Pozwoli to na podjęcie decyzji o wyborze ustawienia DVFS zapewniającego bezpieczną temperaturę pracy lub przeniesieniu wątku z *gorącego* na *zimny* rdzeń z wystarczającym wyprzedzeniem. W komputerach osobistych, z uwagi na konieczność zagwarantowania krótkiego czasu odpowiedzi, stosowane ziarno czasu jest równe 10 ms albo nawet 1 ms [126]. W serwerach i zastosowaniach wysokiej wydajności przydziela się procesom jednorazowo dłuższe okresy, zwiększając tym samym ogólną wydajność całego systemu. Zapewnienie wysokiej wydajności wymusza prognozę na więcej niż jeden okres działania algorytmu szeregowania zadań, a więc 20 ms lub więcej.

W tabeli 4.1 pokazano błędy prognozy temperatury na podstawie ostatnio odczytanej wartości. Błąd ten jest tożsamy z różnicą temperatury pomiędzy dwoma kolejnymi pomiarami. Dane zostały uzyskane podczas pomiarów temperatury procesora testowego w czasie wykonywania testów z zestawu `SPEC CPU 2000` oraz zestawu `PARSEC` z maksymalną liczbą wątków równą 4. Jak widać, podczas przetwarzania pojedynczego wątku już 10 ms wystarczy aby zmienić temperaturę rdzenia o więcej niż 2°C. W przypadku zadań wielowątkowych maksymalna różnica pomiędzy odczytami temperatur co 5 ms wyniosła aż 3.3°C.

4.2 Prognoza temperatury na podstawie aktywności procesora

Do prognozowania temperatury procesora można wykorzystać informacje o uruchomionych na nim zadaniach i ich właściwościach. Model termiczny procesora opisany w rozdziale 3.1 łączy moc rozpraszaną przez poszczególne elementy procesora z ich temperaturą. Odczytując liczbę instrukcji zakończonych w ustalonym czasie można z dużą dokładnością obliczyć moc aktualnie rozpraszaną przez rdzeń procesora. Na podstawie wiedzy o zachowaniu poszczególnych programów w przeszłości można ocenić ich wpływ na temperaturę procesora już w momencie ich uruchamiania. Prognozowanie temperatury tylko na podstawie jej dotychczasowych pomiarów, bez uwzględnienia informacji o wykonywanych programach, ma dwie zasadnicze wady. Po pierwsze, zmiana termicznych warunków wykonywania wątku na danym rdzeniu (np. uruchomienie dodatkowych zadań na procesorze) zmienia jego zachowanie termiczne. Dlatego ekstrapolacja historycznych pomiarów temperatury może prowadzić do błędnych wyników prognozy. Po drugie, w momencie uruchomienia zadania nie jest dostępna informacja historyczna. Obie powyższe wady wynikają z nie uwzględnienia fizycznych właściwości układu.

Temperaturę w czasie t w każdym węźle termicznej sieci RC wyznaczyć można rozwiązując równanie (3.1) [127]:

$$\mathbf{T}(t) = e^{\mathbf{A}(t-t_0)}\mathbf{T}(t_0) + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{BP}(\tau)\,\mathrm{d}\tau. \tag{4.1}$$

Jeśli okres prognozy t_s jest mały moc statyczna nie zmienia się znacznie i można

obliczyć temperaturę w czasie $t = t_0 + t_s$:

$$\mathbf{T}(t) = e^{\mathbf{A}t_s}\mathbf{T}(t_0) + \mathbf{A}^{-1}(e^{\mathbf{A}t_s} - \mathbf{I})\mathbf{B}\mathbf{P}(t_0) \tag{4.2}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową stopnia N_{RC}. Ponieważ macierz stanu $\mathbf{A}$ nie zmienia się, exponentę macierzy $e^{\mathbf{A}t_s}$ można wyliczyć jednorazowo, co znacznie obniża złożoność prognozy temperatury. Aby zachować możliwość prognozy w różnych perspektywach, można obliczyć i zapamiętać exponentę macierzy dla wszystkich rozpatrywanych okresów t_s. Dla każdego t_s można również obliczyć i zapamiętać wartość $\mathbf{A}^{-1}(e^{\mathbf{A}t_s} - \mathbf{I})$. Podstawiając

$$\mathbf{E} = e^{\mathbf{A}t_s} \quad \text{oraz} \quad \mathbf{F} = \mathbf{A}^{-1}(\mathbf{E} - \mathbf{I}), \tag{4.3}$$

otrzymujemy

$$\mathbf{T}(t + t_s) = \mathbf{E}\mathbf{T}(t) + \mathbf{F}\mathbf{P}(t + t_s). \tag{4.4}$$

Stąd wyznaczana jest temperatura systemu po upływie t_s.

Temperatura elementów procesora i jego obudowy, aktualizowana jest na podstawie błędu prognozy. Jeśli suma błędów prognozy wszystkich rdzeni $\sum_i^{N_c} \varepsilon_i$ jest dodatnia, temperatura pozostałych elementów procesora jest zapewne zbyt wysoka, jeśli suma błędów jest ujemna, temperatura radiatora jest zbyt niska. Jest to przesłanką do skorygowania wektora stanu modelu $\mathbf{T}$. Metodą obliczeniowo efektywną jest korekta temperatury radiatora iloczynem sumy błędów i współczynnika k_ε wyznaczonego empirycznie.

Pomiar temperatury otoczenia

Termiczny model procesora zadany równaniem (3.1) opisuje temperaturę każdego węzła termicznej sieci RC względem temperatury otoczenia. Podczas przeprowadzonych eksperymentów zarejestrowano zmiany temperatury powietrza w obudowie komputera związane z wykonywaniem na nim obliczeń, dochodzące do 10°C. Wahać się może także temperatura pomieszczenia, w którym pracuje komputer. Z tego względu celowe jest monitorowanie temperatury otoczenia procesora.

Maksymalna zmiana temperatury w obudowie komputera, zarejestrowana podczas przeprowadzonych eksperymentów, nie przekroczyła 1°C w ciągu 10 s. Można stąd przyjąć, że T_{amb} jest w danym odcinku czasu t_s stała. Jednakże, z powodu wyskalowania czujników temperatury w procesorze względem dozwolonej temperatury

maksymalnej, temperatura otoczenia musi być kontrolowana. Dokładny pomiar można uzyskać umieszczając czujnik temperatury bezpośrednio nad radiatorem, w strumieniu powietrza wlotowego.

Algorytm prognozy

Do prognozowania temperatury wykorzystano opisane w rozdziale 3 modele wiążące wydajność, rozpraszaną moc i temperaturę procesora. Temperatura obliczana jest na podstawie aktualnej wartości odczytanej z czujników zdarzeń umieszczonych standardowo w każdym rdzeniu procesora. Na podstawie równania (3.8) obliczana jest moc rozpraszana przez każdy z rdzeni. Jeśli dany rdzeń pracuje z częstotliwością niższą niż maksymalna, dynamiczna moc skalowana jest według wzoru (3.15). Moc rozpraszana przez każdy z rdzeni procesora jest funkcją jego szybkości s, czyli bieżącego ustawienia napięcia i częstotliwości oraz aktywności rdzenia mierzonej liczbą wykonywanych instrukcji.

Wynikiem przeprowadzonej analizy jest podany niżej algorytm prognozy temperatury oraz wydajności.

Algorytm 1 Algorytm prognozy temperatury $\mathbf{T}$

Wejście: aktualny stan temperatury $\mathbf{T}_p = T_{p,1}, .., T_{p,N_c}$,
aktualna szybkość rdzeni $\mathbf{S}_p = s_{p,1}, .., s_{p,N_c}$,
nowa szybkość rdzeni $\mathbf{S} = s_1, .., s_{N_c}$,
aktywność poszczególnych rdzeni $\mathbf{IPC} = IPC_1, .., IPC_{N_c}$,
wsp. chybień $\mathbf{LLCM} = LLCM_1, .., LLCM_{N_c}$

Wyjście: prognozowana temperatura rdzeni $\mathbf{T}(t + t_s) = T_1, .., T_{N_c}$

1: **dla kolejnych** rdzeni i **wykonaj**
2: odczytaj temperaturę rdzenia T_i
3: wyznacz błąd prognozy $\varepsilon_i = T_{p,i} - T_i$
4: przeskaluj IPC_i względem nowej szybkości rdzenia s_i wg wzoru (3.14)
5: oblicz moc dynamiczną $P_{dyn,i}$ wg wzoru (3.8)
6: przeskaluj moc dynamiczną względem nowej szybkości pracy s_i wg wzoru (3.15)
7: oblicz moc statyczną $P_{stat,i}$ na podstawie wzoru (3.6)

Algorytm 1 Algorytm prognozy temperatury $\mathbf{T}$ – c.d.

8: oblicz moc dynamiczną i statyczną pamięci podręcznych
    na podstawie współczynnika LLCM i temperatury
9: zaktualizuj wektor stanu (temperatura) $\mathbf{T}$ o bieżącą temperaturę rdzeni
10: skoryguj temperaturę radiatora i obudowy $\mathbf{T}[2N_c : N_{RC}]$ o wartość $k_\varepsilon \cdot \sum_{i=1}^{N_c} \varepsilon_i$
11: zaktualizuj wektor wejść (moc) $\mathbf{P}$ o obliczoną moc dynamiczną P_{dyn}
    i statyczną P_{stat} rdzeni i pamięci podręcznych
12: oblicz temperaturę $\mathbf{T}(t + t_s)$ na podstawie wzoru (4.4)
13: zwróć $\mathbf{T}(t + t_s)$

Przed obliczeniem prognozowanej temperatury konieczne jest odczytanie aktualnej temperatury z rejestrów MSR procesora. Temperatura elementów obudowy i fragmentów procesora, dla których bezpośredni pomiar nie jest możliwy nie jest bezpośrednio aktualizowana. Przy każdym wywołaniu algorytmu wyliczany i zapisywany jest błąd prognozy każdego z rdzeni procesora $\varepsilon_i = T_{p,i} - T_i$. Na podstawie sumarycznego błędu prognozy korygowana jest temperatura elementów radiatora. Należy również zaktualizować ustawienia częstotliwości pracy poszczególnych rdzeni w wektorze szybkości rdzeni $\mathbf{S}$.

Przed każdorazowym uruchomieniem algorytmu 1 obliczany jest błąd poprzedniej prognozy. Ponieważ nie jest dostępny mechanizm odczytu temperatury radiatora oraz pamięci podręcznych, do porównania używana jest jedynie temperatura rdzeni. Temperatura radiatora aktualizowana jest na podstawie błędu prognozy.

Aktywność poszczególnych rdzeni wyznaczana jest na podstawie informacji o programach jakie są na nich uruchomione. Podobnie jak w pracy [128] wykorzystywana jest tablica wzorców do prognozowania aktywności. Jednak z uwagi na inny cel prognozy oraz fakt, że wartości IPC oraz LLCM nie są od siebie zależne, w niniejszej pracy zakłada się osobną prognozę liczby wykonanych instrukcji oraz chybień w pamięci podręcznej ostatniego poziomu. Aktywność uruchomionych programów zapisywana jest z użyciem nazwy programu jako klucza. Dla nieznanych programów, pierwszym przybliżeniem aktywności IPC_{start} jest wartość 2. Takie pesymistyczne oszacowanie zapewnia, że prognozowana temperatura nie będzie niższa od rzeczywistej, bo w przypadku większości programów średni poziom IPC jest znacząco niższy niż 2. Proponowane rozwiązanie wymaga zapisywania historii IPC wszystkich programów uruchamianych na komputerze, które działają dostatecznie długo, aby mieć

wpływ na jego temperaturę.

Po aktualizacji temperatury T, szybkości s, aktywności IPC oraz liczby chybień w pamięciach podręcznych ostatniego poziomu $LLCM$ poszczególnych rdzeni, obliczana jest prognozowana wartość mocy statycznej i dynamicznej, rozpraszanej przez każdy z rdzeni oraz pozostałe bloki funkcjonalne procesora. Jeśli częstotliwość pracy rdzenia w nadchodzącym okresie będzie inna niż dotychczasowa, jego prognozowana aktywność i moc dynamiczna skalowane są dodatkowo według równań (3.14) oraz (3.15). Następnie, aktualizowana jest prognozowana moc $\mathbf{P}$ w wektorze wejściowym w części odpowiadającej aktywnym elementom procesora.

Ilość danych koniecznych do zapamiętania nie przekracza kilkudziesięciu kilobajtów i nie stanowi znaczącego obciążenia systemu. Liczba uruchamianych na jednym komputerze programów, które mogą mieć znaczący wpływ na temperaturę procesora jest zwykle nieduża. Dla każdego konieczne jest przechowywanie etykiety/identyfikatora, średniego IPC oraz czasu działania, dla jakiego ta średnia została obliczona. Dodatkowo potrzeba przechowywać informację o zachowaniu programów w postaci tablicy prognoz. Konieczny rozmiar tablicy historii aktywności programu nie przekracza jednego KB jak pokazane zostanie poniżej.

Prognoza IPC i LLCM

Wykorzystanie współczynnika aktywności rdzenia do prognozowania jego temperatury poprawia jakość prognozy w stosunku do metod opartych na historii temperatury. Aktywność rdzenia wyrażona przez IPC programów uruchomionych na nim nie zależy bezpośrednio od temperatury, a jedynie od właściwości wykonywanego programu. Tak więc aktywność rdzenia nie zmienia się w zależności od zmian warunków termicznych, w szczególności temperatury sąsiednich rdzeni i otoczenia.

Prognozę mocy lub aktywności procesora można uzyskać na różnymi metodami. W pracy [57] do prognozy temperatury wykorzystano ostatnią wartość mocy. Z uwagi na dużą zmienność zachowania niektórych programów skutkowało to błędami prognozy mocy sięgającymi 40%. Do prognozy IPC oraz LLCM, a ogólniej fazy programów, wykorzystać można również predyktor stochastyczny RLE (ang. *Marcov Run Length Encoding Predictor*) [129] lub prognozę na podstawie tablicy historii [128]. Działanie takiej tablicy przypomina działanie globalnego bufora prognozy rozgałęzień stosowanego w procesorach, który bazuje na tablicy historycznych wzorców wykonania skoku

oraz prognozy przypisanej do każdego z nich.

Z podobnej tablicy historii skorzystano w niniejszej pracy. W odróżnieniu od dużej globalnej tablicy wykorzystywanej w pracy [130] tu każdemu programowi przyporządkowane są dwie lokalne tablice prognozy aktywności: tablica instrukcji na cykl zegara (LTP_{IPC}) oraz chybień w pamięci podręcznej ostatniego poziomu (LTP_{LLCM}). Wykorzystanie globalnej tablicy dla wszystkich programów powoduje dwa problemy. Po pierwsze wykonywanie operacji na dużej globalnej tablicy wymaga większej liczby obliczeń przy każdej operacji aktualizacji i wyszukiwania trafienia. Ponadto możliwe jest dopasowanie do aktualnego wzorca sekwencji współczynników IPC które dotyczą innego programu i tym samym uzyskanie błędnej prognozy.

Predyktor oparty na tablicy historii składa się z dwóch podstawowych elementów: wektora zawierającego wzorzec – przedziały w których znalazło się h ostatnich odczytów, oraz tablicy mieszczącej l wzorców, przypisaną każdemu prognozowaną wartość oraz wskaźnik poprawności danej prognozy. Wskaźnik mniejszy od 0 oznacza wpis nieważny. Działanie tablicy prognozy zostało opisane w algorytmie 2.

Algorytm 2 Algorytm prognozy fazy IPC

Wejście: aktywność rdzenia IPC_i
tablica LTP_{IPC} programu przetwarzanego na rdzeniu
modyfikator trafienia m_h i chybienia m_m

Wyjście: prognozowana aktywność rdzenia IPC_{i+1}

1: ustaw $IPC_{i+1} = IPC_i$
2: odejmij 1 od wskaźnika poprawności każdego wiersza w tablicy
3: znajdź listę trafień $\mathcal{T}$ bieżącego wzorca w tablicy
4: **jeśli** $\mathcal{T} = \emptyset$ **wtedy**
5: zastąp najdawniej używany wpis w tablicy bieżącą wartością historii oraz IPC_i
6: **w przeciwnym razie**
7: **dla kolejnych** indeksów **w** $\mathcal{T}$ **wykonaj**
8: **jeśli** IPC_i równe jest prognozie w tablicy pod danym indeksem **wtedy**
9: zwiększ wskaźnik poprawności o m_h
10: **w przeciwnym razie**
11: zmniejsz wskaźnik poprawności o m_m

Algorytm 2 Algorytm prognozy fazy IPC –

zaktualizuj wzorzec o bieżącą wartość IPC_i i odrzuć najdawniejszą wartość
znajdź w tablicy LTP wpis odpowiadający zaktualizowanemu wzorcowi
jeśli wpis istnieje **wtedy**
    przypisz prognozę do IPC_{i+1}
zwróć IPC_{i+1}

Składa się on z dwóch zasadniczych etapów. Po odczytaniu bieżącej wartości współczynnika aktualizowana jest tablica prognozy. Wskaźnik poprawności w każdym wierszu obniżany jest o jeden. Dodatkowo wskaźnik aktualizowany jest w każdym wierszu który pasował do poprzedniego wzorca. Jeśli zapisana prognoza była poprawna (równa bieżącej wartości) wskaźnik powiększany jest o modyfikator, którego wartość dobrana została empirycznie. W razie błędnej prognozy, wskaźnik poprawności jest obniżany, jeśli w tablicy brak pasującego wzorca jest on wstawiany, wraz z bieżącą wartością jako prognozą, w miejsce wpisu o najniższym wskaźniku poprawności. Po aktualizacji tablicy prognozy, następuje aktualizacja wzorca historii o bieżącą wartość, odrzucenie najdawniejszej i wyszukanie w tablicy pasującego wiersza. W razie nietrafienia prognozowana jest obecna wartość, a w przypadku wielokrotnego trafienia w tablicy – prognoza o najwyższym wskaźniku poprawności.

W praktycznej implementacji algorytm można wykonać jednokrotnie przeglądając tablicę, co skutkuje złożonością $\mathrm{O}(h \cdot l)$. W przeprowadzonych eksperymentach najwyższą skuteczność miała tablica zawierająca 64 wiersze i wzorce o długości 8 okresów, co przekłada się na umiarkowany rozmiar tablicy (<1 KB).

W rzadkich przypadkach programów, dla których występują błędne prognozy i skuteczność predyktora opartego na tablicy jest gorsza niż prognozy ostatniej wartości, zaimplementowany algorytm zwraca ostatnią wartość. Aby umożliwić dopasowanie wzorca do historii w tablicy, za równe przyjęto współczynniki IPC (lub LLCM) nie różniące się o więcej ustalona z góry wartość progowa. Ważne jest także skuteczne eliminowanie z tablicy wpisów z błędną prognozą, dlatego też modyfikator chybienia m_m musi być wielokrotnie większy niż modyfikator trafienia m_h. Konkretne wartości obu modyfikatorów zależą między innymi od rozmiaru tablicy prognozy.

Zbiorcze wyniki prognozy wykorzystującej tablicę 64 wzorców o długości 8 kroków i porównanie z prognozą na podstawie ostatniej wartości przedstawiono w tabeli 4.2. Można zauważyć, że zarówno błąd średniokwadratowy (MSE) jak i liczba okresów

Tabela 4.2: Skuteczność prognozy IPC dla programów z pakietów `PARSEC` i `SPEC`, $t_s = 100$ ms

		tablica 64x8		ostatnia wartość	
test	l. krok.	MSE	$\lvert\mathbb{E}\rvert$	MSE	$\lvert\mathbb{E}\rvert$
`SPEC_int`	8000	296.6	907	313.2	933
`SPEC_fp`	12738	1031.1	2061	1467.2	3225
`PARSEC`	46000	385.6	1488	549.9	1881

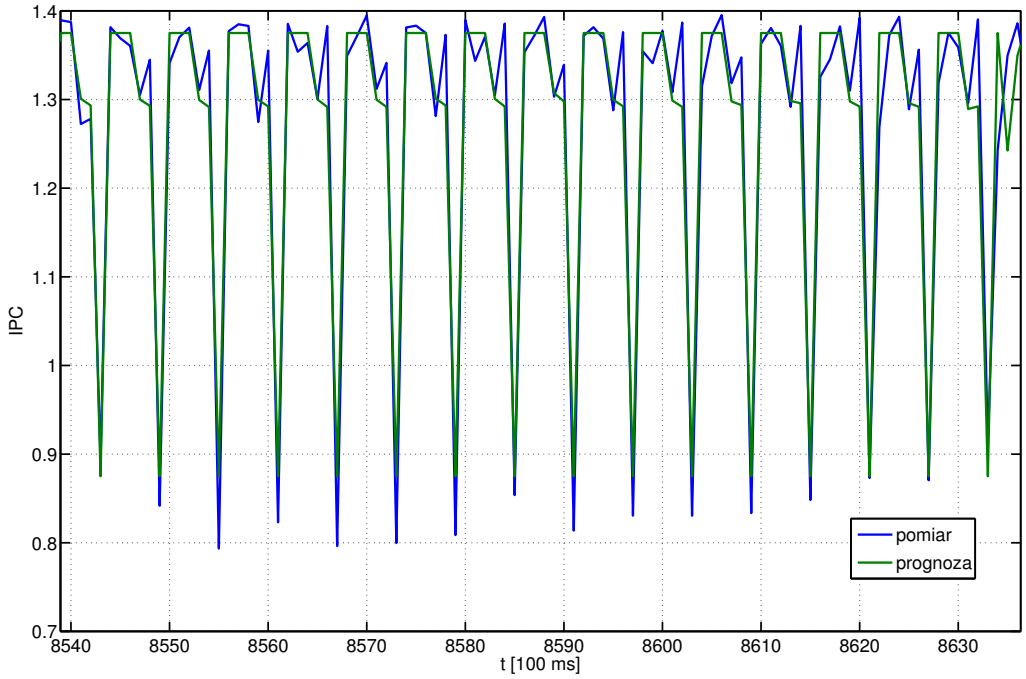
z błędną prognozą ($|\mathbb{E}|$) uległa zmniejszeniu w przypadku wszystkich testowanych zestawów programów. Najmniejsza poprawa w stosunku do prognozy ostatniej wartości widoczna jest w przypadku programów stałoprzecinkowych `SPEC int`, które wykazują najmniejszą powtarzalność zachowania.

Inaczej niż w pracy [128], tu potrzebna jest osobna prognoza współczynników IPC oraz LLCM. Zgodnie z równaniem (3.13) ograniczanie częstotliwości pracy rdzenia z wysokim współczynnikiem chybień w pamięci podręcznej ostatniego poziomu skutkuje relatywnie mniejszym spowolnieniem pracy. Aby ograniczyć efektywnie zużycie energii, faktycznie wystarczy znać stosunek LLCM/IPC, ale oba parametry są słabo powiązane, to znaczy wysoki współczynnik LLCM nie implikuje bezpośrednio niskiej średniej liczby instrukcji wykonanych w cyklu zegara.

Przykładowy wynik prognozy oraz fragment rzeczywistego przebiegu IPC przedstawiono na rysunku 4.2. Można zauważyć wysoką skuteczność prognozy przebiegu o dużej zmienności. Jednym z parametrów, które mają największy wpływ na skuteczność prognozy jest liczba przedziałów na które dzielony jest cały zakres prognozowanej wartości. Im więcej przedziałów, tym dokładniejszy wynik prognozy, ale mniejsza liczba trafień w tablicy. Dla przeanalizowanych przebiegów IPC najlepsze wyniki uzyskano dzieląc cały zakres obserwowanych wartości IPC (od 0 do 2.5) na 10 odcinków. Niestety, wiele programów nie ma okresowego charakteru i prognoza inna niż na podstawie ostatnio obserwowanej wartości będzie wtedy nieskuteczna.

4.2.1 Aktualizacja modelu termicznego

Zmieniające się warunki otoczenia mogą wymuszać aktualizację modelu termicznego procesora. Spowodowane może to być zmianą temperatury otoczenia lub zmianą



Rysunek 4.2: Wynik prognozy IPC dla programu `lucas`, $ts = 100\,ms$, $n = 4$

szybkości obrotów wentylatora umieszczonego na radiatorze. W obu przypadkach następuje zmiana rezystancji termicznej pomiędzy radiatorem a temperaturą otoczenia. Z uwagi na dużą zmienność warunków otoczenia procesora, aktualizacja modelu termicznego jest szczególnie pożądana w komputerach przenośnych.

Poszczególne parametry modelu odpowiadają właściwościom fizycznym układu składającego się z procesora, obudowy, radiatora i jego otoczenia. Dzięki temu rozszerzenie opisanego powyżej algorytmu prognozy o dynamiczną aktualizację modelu termicznego nie powinno nastręczać trudności. Na podstawie pomiarów temperatury otoczenia procesora (np. powietrza wlotowego do radiatora) oraz pomiaru obrotów wiatraka (po uprzednim określeniu zależności rezystancji termicznej od prędkości obrotowej wiatraka) można zaktualizować model termiczny.

Macierze stanu $\mathbf{A}$ i wejść $\mathbf{B}$ określające dynamiczne właściwości termiczne procesora, a opisane w rozdziale 3 można wyznaczyć na podstawie pojemności cieplnej poszczególnych składników modelu oraz rezystancji termicznej pomiędzy nimi [34].

Pojemność cieplną i-tego elementu skupionego modelu procesora oznaczymy jako c_i. Rezystancja pomiędzy i-tym, a j-tym elementem obwodu termicznego oznaczymy jako $r_{i,j}$. Oczywiście, $r_{i,j} = \infty$ dla bloków b_i i b_j, które się nie sąsiadują ze sobą.

Tabela 4.3: Czas potrzebny na obliczenie prognozy temperatury w zależności od liczby węzłów modelu N_{RC}

N_{RC}:	4	7	13	25	49	97	193
czas [μs]:	2.4	2.7	2.9	3.6	6.5	17.4	73.5

Macierz $\mathbf{D} = (d_{i,j})_{N_{RC} \times N_{RC}}$ jest macierzą konduktancji termicznej:

$$d_{i,j} = \begin{cases} -\sum_{k=1}^{N} \frac{1}{r_{i,k}} & \text{if } i = j \\ \frac{1}{r_{i,j}} & \text{if } i \neq j \end{cases} \tag{4.5}$$

Po utworzeniu macierzy $\mathbf{D}$, od jej elementów przekątniowych odpowiadających fragmentom radiatora, przez które odprowadzane jest na zewnątrz układu ciepło należy odjąć odwrotność rezystancji termicznej radiatora, proporcjonalnie do ich powierzchni wymiany ciepła. Macierz $\mathbf{C}$ jest macierzą pojemności termicznej:

$$c_{i,j} = \begin{cases} c_{i,j} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4.6}$$

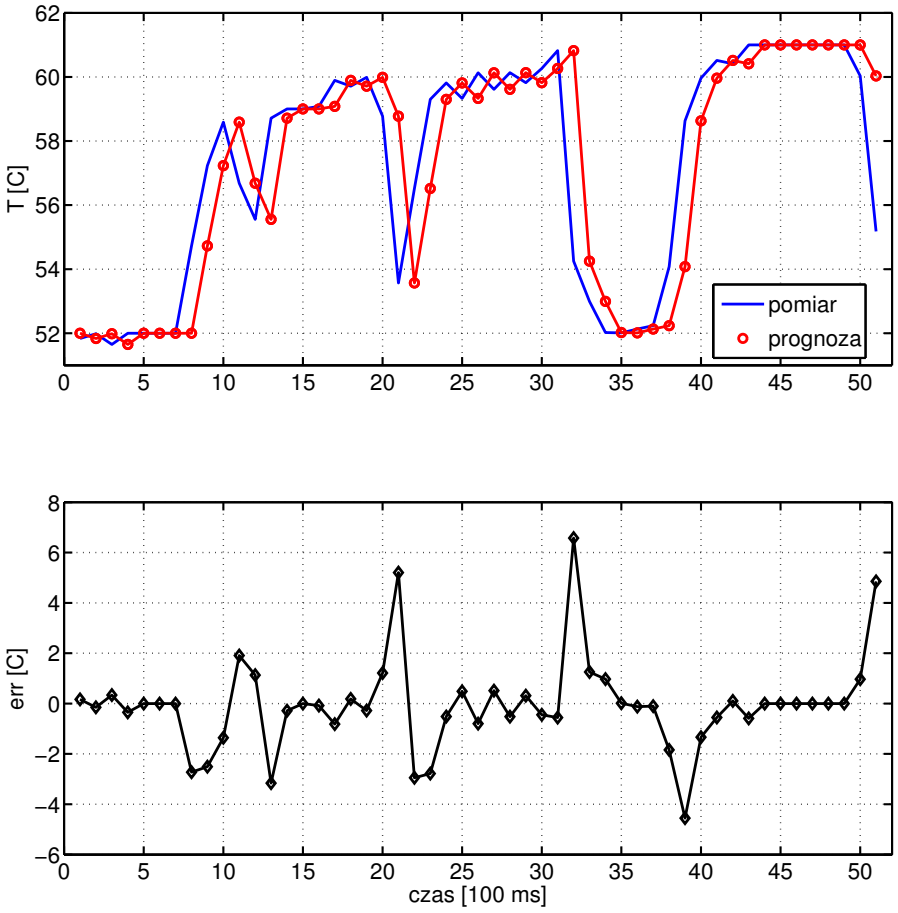
Na podstawie macierzy $\mathbf{C}$ i $\mathbf{D}$ wyznaczane są macierze $\mathbf{A}$ oraz $\mathbf{B}$ z równania stanu (3.1)

$$\mathbf{A} = \mathbf{C}^{-1}\mathbf{D} \quad \text{oraz} \quad \mathbf{B} = \mathbf{C}^{-1}. \tag{4.7}$$

Aktualizacja modelu z powodu zmiany rezystancji termicznej radiatora wymaga jedynie zmiany macierzy $\mathbf{D}$ oraz przeliczenia macierzy $\mathbf{A}$ i $\mathbf{B}$. Koszt obliczeniowy takiej operacji jest umiarkowany – wymagane jest jedynie wykonanie mnożenia macierzy albo wstępna kalkulacja macierzy dla każdej dostępnej prędkości obrotów wentylatora.

4.3 Porównanie ze znanymi rozwiązaniami

W implementacji algorytmu prognozy wykorzystano program `HotSpot` [28], który na podstawie parametrów procesora określonych w rozdziale 3.3 wylicza macierze konduktancji $\mathbf{D}$ i pojemności termicznej procesora $\mathbf{C}$. Na podstawie tych macierzy budowany jest model termiczny. Czas potrzebny na obliczenie prognozowanej temperatury zależy od implementacji, dokładności modelu termicznego (liczba węzłów N_{RC} w obwodzie termicznym) oraz częstotliwości aktualizacji prognozy. Tabela 4.3



Rysunek 4.3: Prognoza temperatury na podstawie ostatniej wartości

zawiera czasy prognozy zmierzone na podstawie implementacji w środowisku MATLAB w zależności od rozmiarów modelu termicznego. Można przyjąć, że zoptymalizowana implementacja w języku C powinna być zdecydowania szybsza. Tak więc czasy prognozy nawet dla procesorów wyposażonych w wiele czujników temperatury są pomijalne, biorąc pod uwagę czas pojedynczego kroku algorytmu zarządzającego działaniem procesora (>10 ms).

W celu oceny skuteczności algorytmu prognozy temperatury procesora na podstawie aktywności rdzeni zebrano profile temperatury podczas wykonywania programów sekwencyjnych i wielowątkowych na komputerze testowym. W każdym profilu zapisane zostały informacje o temperaturze każdego z rdzeni, liczbie zakończonych instrukcji, chybień w pamięciach podręcznych oraz temperaturze otoczenia T_{amb}.

Najprostszym algorytmem jest prognoza ostatniej wartości. Przy założeniu, że zmiany temperatury są powolne w stosunku do częstości aktualizacji prognozy, róż-

nica między prognozowaną a rzeczywistą temperaturą nie jest duża. Niestety temperatura w procesorze może zmieniać się nawet w tempie $1°\mathrm{C}/ms$ [123] co wyklucza prognozę długoterminową na podstawie ostatniej wartości. Z tego powodu konieczne jest zapewnienie wystarczającego dystansu pomiędzy aktualną temperaturą, a maksymalną $T_{\max}$, co może powodować ograniczenie wydajności. Przykładowy wynik prognozy opartej na ostatniej wartości zilustrowano na rysunku 4.3, a wyniki prognozy dla zestawów programów testowych zaprezentowano w tabeli 4.1.

Inny prosty mechanizm prognozy temperatury odwołuje się do założenia, że aktualny trend zmiany temperatury w każdym punkcie procesora utrzyma się w następnym okresie czasu. Jeśli temperatura rośnie i zakładamy kontynuację tego trendu, prognozowany jest dalszy wzrost temperatury i zastosowany zostaje mechanizm skutkujący jej obniżeniem. W najprostszej formie taka prognoza temperatury może być opisana jako:

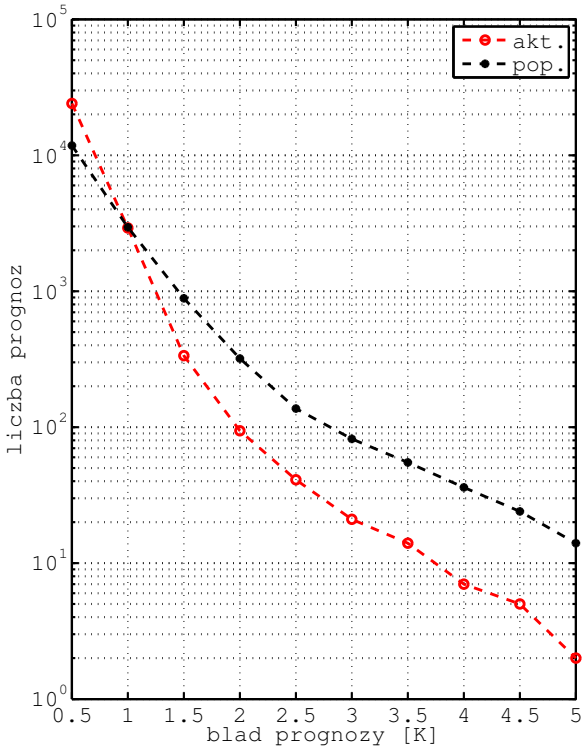
$$T(t_0 + t_s) = T(t_0) + (T(t_0) - T(t_0 - t_s)) \,. \tag{4.8}$$

W rozszerzonej formie, prognoza na podstawie n ostatnich pomiarów temperatury może być przedstawiona jako funkcja n ostatnich pomiarów temperatury.

$$T(t_0 + t_s) = T(t_0) + f(T_{0-n}, .., T_0) \tag{4.9}$$

Taka metoda prognozy jest całkowicie nieskuteczna w przypadku programów, podczas przetwarzania których temperatura szybko się zmienia (oscyluje). Prognoza temperatury na podstawie chwilowego trendu może być użyta do ostrzegania przed przekroczeniem temperatury maksymalnej [131]. Podobnie, wykorzystanie regresji liniowej wartości liczników zdarzeń może być wykorzystane do przewidywania wystąpienia gorących punktów [66]. Temperatura pojedynczego rdzenia zależy nie tylko od jego aktywności, ale również od warunków otoczenia i mocy rozpraszanej przez cały procesor. Z tego powodu proste metody prognozy gorących punktów bezpośrednio na podstawie odczytów liczników zdarzeń skutkują niską dokładnością. Za ich stosowaniem przemawia minimalna złożoność obliczeniowa.

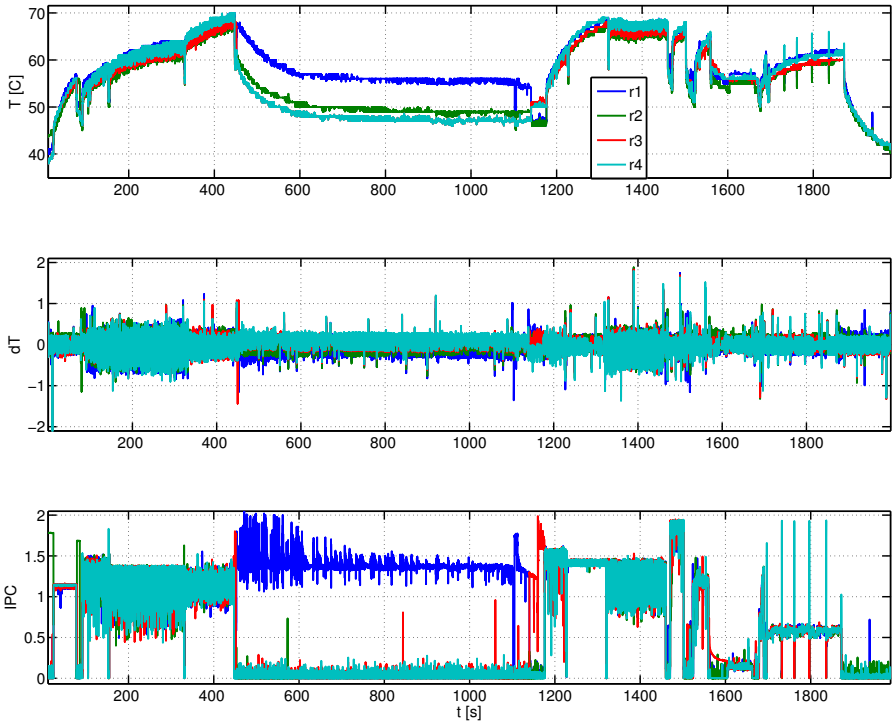
W innej znanej metodzie prognozy temperatury procesora wielordzeniowego wykorzystuje się model ARMA [64]. Jest on skuteczny przy założeniu, że modelowany proces jest stacjonarnym procesem stochastycznym i w danych występuje autokorelacja. Wadą metody prognozy jest konieczność bieżącego monitorowania jej skuteczności, a także występowanie fazy identyfikacji i estymacji parametrów modelu.



Rysunek 4.4: Liczba błędnych prognoz w funkcji wielkości błędu dla prezentowanego algorytmu (akt) i prognozy na podstawie poprzedniej wartości (pop) przy $t_s = 100\,\text{ms}$

Poszczególne metody prognozy temperatury zostały porównane na podstawie implementacji w środowisku `MATLAB` z wykorzystaniem profili zapisanych na komputerze testowym podczas wykonywania programów z pakietów `SPEC CPU 2000` i `PARSEC`. Liczbę prognoz z błędem większym niż 2°C i maksymalny błąd prognozy przedstawiono w tabeli 4.4. Prognoza na podstawie aktywności procesora (**akt. tablica**) została wykonana za pomocą algorytmu 1. Żeby oszacować wpływ błędnej oceny zachowania programów na dokładność prognozy, przeprowadzono również pomiary zakładając doskonałą prognozę fazy programów (**akt. wyrocznia**) i prognozę na podstawie ostatniej wartości aktywności (**akt. ost. wart**). Przewaga proponowanej metody jest tym wyraźniejsza im dłuższy jest okres prognozy.

Na rysunku 4.4 przedstawiono liczbę błędnych prognoz w funkcji wielkości błędu dla prognozy ostatniej wartości i proponowanego algorytmu prognozy. Można zauważyć, że prezentowana metoda skutkuje znacznym obniżeniem liczby błędnych prognoz. Tylko w przypadku błędów mniejszych niż 0,5°C prognoza na podstawie poprzedniej wartości daje lepsze rezultaty.



Rysunek 4.5: Rzeczywisty przebieg temperatury, błąd prognozy δT i aktywność wszystkich rdzeni procesora testowego podczas wykonywania testów z zestawu `PARSEC`

Przebieg temperatury, błąd prognozy oraz wartości współczynnika IPC poszczególnych rdzeni podczas wykonania całego zestawu testów `PARSEC` przedstawiono na rysunku 4.5. Przedstawione wyniki uzyskano dla czasu prognozy t_s równego 20 ms. Można zauważyć, że większość błędów prognozy jest mniejsza niż 1°C.

4.4 Wnioski

Jakość prognozy wyraźnie zależy od dokładności przyjętego modelu termicznego. Do jego stworzenia niezbędna jest znajomość topografii procesora. Dla przykładu, testowy procesor składa się z 2 modułów umieszczonych w jednej obudowie. Każdy z nich zawiera 2 rdzenie ze wspólną pamięcią podręczną drugiego poziomu. Podział ten przekłada się bezpośrednio na bardzo wysoką rezystancję termiczną pomiędzy częściami procesora leżącymi na różnych kawałkach krzemu. Uwzględnienie tego podziału poprawiło dokładność prognozy.

Konduktancja termiczna w pionie (wg orientacji na rysunku 3.3) dominuje nad

Tabela 4.4: Liczba prognoz z błędem większym niż 2°C i maksymalny błąd prognozy temperatury uzyskanej różnymi metodami

	$\lvert\mathbb{E}_2\rvert$				$\varepsilon_{\max}$			
metoda	10 ms	20 ms	40 ms	100 ms	10 ms	20 ms	40 ms	100 ms
ostatnia	4	4	310	320	2.88	2.88	9.29	7.7
akt. ost. wart.	4	4	36	123	2.84	2.81	3.47	6.50
akt. tablica	4	4	29	112	2.83	2.85	3.22	6.17
akt. wyrocznia	4	4	23	94	2.92	2.82	3.00	5.99

konduktancją pomiędzy blokami funkcjonalnymi procesora pod względem wpływu na dynamiczne zachowanie termiczne procesora. Rezystancja pionowa jest kilkudziesięciokrotnie niższa niż pozioma. Ponadto, z uwagi na masę oraz materiał o wyższej pojemności cieplnej niż w przypadku rdzenia procesora (krzem), pojemność cieplna radiatora jest o 3 rzędy większa.

Zastosowany model termiczny bazuje na analogii pomiędzy zjawiskami termicznymi, a elektrycznymi. Węzłom modelowanego termicznego obwodu RC przypisana jest temperatura względem temperatury otoczenia. Natomiast, czujniki temperatury w procesorze podają wskazania w skali bezwzględnej (jako różnicę względem temperatury maksymalnej). Z tego względu, zastosowanie powyższego modelu termicznego wymaga monitorowania temperatury w obudowie komputera. Temperatura otoczenia zmienia się relatywnie powoli, względem temperatury samego procesora i jej pomiary mogą być wykonywane dużo rzadziej niż odczyty temperatury samego procesora.

Rozdział 5

Sterowanie wydajnością procesora z ograniczeniem temperatury

5.1 Skuteczność mechanizmów DTM

Poza wynikami pomiarów temperatury, kluczową miarą skuteczności mechanizmów DTM jest czas przetwarzania zadań. Metryki oceniające wydajność na podstawie faktycznie wykonanej pracy są dużo dokładniejsze niż odwołujące się do aktywności procesora. Użycie IPC jako metryki wydajności również może prowadzić do błędnych konkluzji [90]. Ponieważ program może wykonywać instrukcje podczas gorącego czekania na zasoby, lub oczekiwać np. na obsługę chybienia w TLB, liczba wykonanych instrukcji nie jest stała dla danego programu i zależy od (mikro)architektury procesora oraz, w ogólności, od systemu na jakim jest on wykonywany. Na poziomie systemowym, zmiana kolejności wykonania wpływa na przydział zasobów dla wszystkich zadań. Ma to znaczenie szczególnie w przypadku aplikacji wielowątkowych, gdzie synchronizacja między wątkami wpływa na czas zakończenia całego zadania. Z tego powodu, w poniższym rozdziale, wydajność będzie oceniana na podstawie czasu potrzebnego na ukończenie zadania.

Niedeterminizm wyników

Z uwagi na przeprowadzanie eksperymentów na wielozadaniowym systemie, czas wykonania zadania jest w pewnym stopniu niedeterministyczny. Podlega on losowym zmianom ze względu na procesy pracujące w tle. W celu ustalenia możliwych zmian

Tabela 5.1: Minimalny i maksymalny czas wykonania testów z pakietu PARSEC oraz procentowa różnica różnica między nimi dla 2 i 4 wątków

	n=2 wątki			n=4 wątki		
benchmark	t_{min} [s]	t_{max} [s]	Δt [%]	t_{min} [s]	t_{max} [s]	Δt [%]
blackscholes	136.07	140.61	4.40	81.81	82.48	1.09
bodytrack	126.66	127.91	1.31	68.41	69.05	1.24
canneal	174.20	175.07	0.66	119.97	120.22	0.28
dedup	28.51	29.40	4.10	21.81	25.04	18.75
facesim	304.21	306.01	0.79	190.57	193.15	1.79
ferret	252.33	252.78	0.24	130.46	130.50	0.04
fluidanimate	263.66	264.00	0.17	145.72	146.52	0.73
freqmine	723.77	725.71	0.36	723.95	728.12	0.77
streamcluster	257.56	258.01	0.23	179.19	179.86	0.50
swaptions	204.78	204.94	0.10	103.50	103.69	0.24
vips	52.97	53.01	0.07	41.99	42.30	0.98
x264	53.26	53.83	1.07	41.72	42.52	2.54
$\sum$	2633.19	2675.81	1.61	1849.10	1863.45	0.77

czasu wykonywania programów, zmierzono czas wykonania wszystkich programów z zestawu PARSEC. Każdy z programów został uruchomiony czterokrotnie. Zanotowano minimalny i maksymalny czas oraz różnicę między nimi podzieloną przez średni czas. Eksperyment ten przeprowadzono dla każdego programu zrównoleglonego na dwa i cztery wątki. Uzyskane wyniki przedstawiono w tabeli 5.1. W przypadku prawie wszystkich programów, których czas wykonywania był dłuższy niż 100 sekund zmierzone czasy nie różniły się o więcej niż 1 procent, a dla całego zestawu testów różnica ta nie przekraczała 2 procent. Oznacza to, że wyniki są powtarzalne i dla dłuższych testów pojedynczy pomiar powinien być miarodajny.

5.1.1 Dynamiczne skalowanie napięcia i częstotliwości

Poniżej przedstawione zostaną wyniki oceny skuteczności dynamicznego skalowania napięcia i częstotliwości jako mechanizmu kontroli temperatury procesora oraz eksperymentalnego określenia narzutów w postaci spowolnienia wykonywania zadań

wynikających z konieczności ograniczenia temperatury. W literaturze dostępnych jest niewiele prac bezpośrednio poświęconych ocenie skuteczności DVFS. Jednym z przykładów jest praca [132], w której przeanalizowano wpływ skalowania napięcia na temperaturę komputera z procesorem `Intel Pentium M`.

Skalowanie napięcia i częstotliwości jest skutecznym mechanizmem obniżania mocy rozpraszanej, co skutkuje obniżeniem temperatury. Wskazaniem do ograniczenia częstotliwości pracy rdzenia jest niska aktywność (niski współczynnik IPC) przetwarzanego na nim wątku. Jak pokazano w rozdziale 3.6, wydajność wątków o niskim IPC i dużej liczbie chybień w pamięciach podręcznych zmniejsza się wolno wraz z redukcją częstotliwości pracy rdzenia. Praktycznie jedyną możliwością ograniczenia temperatury procesora, kiedy wszystkie rdzenie są obciążone, jest ograniczanie częstotliwości pracy. Niestety, nie zawsze obniżenie częstotliwości i napięcia danego rdzenia wystarcza do obniżenia jego temperatury do akceptowalnego poziomu. W takich sytuacjach konieczne jest spowolnienie pracy sąsiednich rdzeni, albo całkowite czasowe wstrzymanie przetwarzania na przegrzewającym się rdzeniu. Konieczność zastosowanie alternatywnego względem DVFS mechanizmu może być spowodowana potrzebą zapewnienia wymaganej wydajności. W przypadku wątku o wysokim priorytecie lepszym rozwiązaniem może być ograniczenie temperatury całego procesora poprzez ograniczenia szybkości działania pozostałych rdzeni. DVFS może być uzupełniony o migrację wątków, która będzie skuteczna w określonych okolicznościach. Przede wszystkim w przypadku wątków o wysokim IPC oraz w sytuacji gdy obciążona jest mała liczba spośród dostępnych rdzeni. Kolejnym przeciwwskazaniem do zastosowania DVFS jest przegrzewanie się rdzenia wielowątkowego (SMT), jeśli uruchomiono na nim więcej niż jeden wątek. W takiej sytuacji skuteczniejszą metodą może się okazać przeniesienie jednego z wątków na inny rdzeń.

Czas zmiany poziomu napięcia i częstotliwości jest we współczesnych procesorach pomijalny z punktu widzenia zarządzania temperaturą i wynosi maksymalnie 10 μs. Stanowi to nie więcej niż 1% czasu między kolejnymi przerwaniami zegara systemowego [126]. Dzięki temu skalowanie napięcia i częstotliwości można stosować doraźnie – jako mechanizm zabezpieczający przed przegrzewaniem rdzeni. Osobną kwestią jest niedopasowanie częstotliwości pracy rdzenia do aktualnych ograniczeń temperatury i wymagań co do oczekiwanej wydajności.

Pierwszym elementem oceny skuteczności DVFS w ograniczaniu temperatury jest uruchomienie programów testowych na procesorze z ustalonym poziomem napięcia

Tabela 5.2: Czas przetwarzania programów testowych oraz maksymalna zarejestrowana temperatura, względem temperatury otoczenia, w funkcji częstotliwości pracy procesora

	2.00 GHz		2.33 GHz		2.66 GHz	
benchmark	t [s]	T_{max} [K]	t [s]	T_{max} [K]	t [s]	T_{max} [K]
SPEC int	264,55	17,81	227,63	24,12	201,40	29,13
SPEC fp	745,23	17,62	645,98	22,94	571,25	29,31
bodytrack n=2	167,34	15,50	143,08	19,31	125,05	24,11
facesim n=4	237,30	21,52	206,06	27,88	191,42	35,48

i częstotliwości. W tabeli 5.2 przestawiono wyniki pomiaru czasu przetwarzania różnych zestawów programów testowych oraz maksymalną zarejestrowaną temperaturę. Temperatura podana została względem temperatury otoczenia procesora aby zminimalizować wpływ czynników zewnętrznych na wynik eksperymentu. Żeby ograniczyć wpływ zaszeregowania wątków na różne rdzenie, programom z pakietu `SPEC CPU` został przypisany jeden rdzeń przy użyciu programu `taskset`. Jak widać, obniżenie szybkości pracy procesora z 2.66 GHz do 2.00 GHz pozwoliło ograniczyć maksymalną temperaturę o ponad 11 K w przypadku programów sekwencyjnych (`SPEC`). W przypadku programu `facesim` uruchomionego z 4 wątkami ta różnica sięga prawie 14 K. Oczywiście, zmniejszenie temperatury wiąże się z wydłużeniem czasu przetwarzania o 24 do 34%.

Ocenę skuteczności mechanizmu DVFS do ograniczania temperatury w procesorze testowym przeprowadzono w następujący sposób: została ustalona temperatura maksymalna T_{max}, taka aby przy maksymalnym poziomie DVFS, znaczącą część czasu wykonania programów testowych, procesor przekraczał T_{max}. Uruchomiony został pełen zestaw programów `PARSEC` oraz `SPEC`. Programy wielowątkowe z pakietu `PARSEC` były uruchamiane pojedynczo, a programy z pakietu `SPEC CPU` grupami po 4, tak aby dostatecznie obciążyć komputer. Po zakończeniu działania danej czwórki, uruchamiane były kolejne 4 programy, aż do wyczerpania zestawu aplikacji stało i zmiennoprzecinkowych. Co 100 ms podstawowy algorytm DVFS (3) ustawiał napięcia i częstotliwości przypisane poszczególnym rdzeniom, tak aby ograniczyć temperaturę a jednocześnie nie ograniczać wydajności, jeśli temperatura pozostawała na bezpiecznym poziomie. Prosty algorytm DVFS (3), na podstawie aktualnej temperatury i szybkości poszcze-

Algorytm 3 Podstawowy algorytm DVFS

Wejście: aktualna szybkość $\mathbf{S_0} = s_1, .., s_n$,

1: dostępne szybkości $\mathbf{S_a} = s_1, .., s_m$,

2: temperatura rdzeni procesora $\mathbf{T_0} = T_1, .., T_n$,

3: temperatura maksymalna T_{max} i progowa T_{th}

Wyjście: ustawienie szybkości rdzeni procesora $\mathbf{S}_{t_s} = s_1, .., s_n$

4: **dla kolejnych** $i = 1$ do N_c **wykonaj**

5: **jeśli** $T_i \geqslant T_{max}$ **wtedy**

6: $s_i = \mathbf{S_a}[1]$

7: **w przeciwnym razie jeśli** $T_i < T_{th}$ **wtedy**

8: **jeśli** $s_i < \mathbf{S_a}[m]$ **wtedy**

9: $idx = znajdź(s_i, S_a)$

10: $s_i = S_a[idx + 1]$

gólnych rdzeni zwraca ustawienia częstotliwości na kolejny okres. Co ustalony czas Δt odczytywana jest temperatura wszystkich rdzeni. Jeśli temperatura danego rdzenia przekracza wartość maksymalną, ustawiana jest minimalna dostępna szybkość. Jeśli temperatura jest niższa niż temperatura progowa, szybkość procesora jest zwiększana o jeden poziom. Temperatura progowa T_{th} musi być dobrana tak, żeby mechanizm DTM nie wpadał w oscylacje, jednocześnie nie ograniczając nadmiernie wydajności. Konkretna wartość temperatury progowej zależy również od szumu czujników oraz narzutu związanego z mechanizmem DTM.

Ponieważ w procesorze wykorzystywanym do testów zakres ustawień DVFS jest ograniczony do 75% częstotliwości maksymalnej $f_{\max}$, zmniejszenie szybkości pojedynczego rdzenia może nie być wystarczające do kontrolowania temperatury. Z tego powodu zaimplementowano dodatkowy algorytm (oznaczony jako *agresywny DVFS*), który ogranicza również szybkość sąsiednich, względem przegrzewającego się, rdzeni.

Wyniki pomiarów temperatury i czasu wykonania programów testowych przedstawiono w tabeli 5.3. Temperatura maksymalna $T_{\max}$ i progowa T_{th} zostały ustalone na poziomie 60 i 57°C. Można zauważyć, że reaktywne algorytmy są niewystarczające do utrzymania temperatury poniżej zadanego poziomu. Pomiędzy kolejnymi wywołaniami algorytmu temperatura może wzrosnąć o kilka stopni. Zastosowanie agresywnego algorytmu powoduje zwiększenie czasu przetwarzania zadań (maksymalnie o 24% w przeprowadzonych eksperymentach), jednocześnie ograniczając czas, w którym temperatura przekracza dozwoloną wartość.

Tabela 5.3: Czas działania, temperatura maksymalna, liczba okresów w których temperatura przekraczała T_{max} dla wybranych programów z zestawów `SPEC CPU 2000` i `PARSEC` w zależności od algorytmu DVFS

	prosty DVFS			**agresywny DVFS**		
benchmark	t [s]	max T [C]	T$\geqslant T_{\max}$	t [s]	max T [C]	T$\geqslant T$ max
SPEC int	376	63	411	382	63	224
SPEC fp	350	63	484	400	62	177
blackscholes	86	62	100	89	59	0
bodytrack	74	62	171	80	61	14
canneal	130	57	0	131	57	0
dedup	23	58	0	29	59	0
facesim	222	64	492	227	62	124
ferret	140	63	109	149	62	20
fluidanimate	176	64	210	183	62	121
freqmine	726	58	0	738	61	1
streamcluster	202	66	578	204	64	12
swaptions	123	64	308	129	63	69
vips	51	63	70	63	62	22
x264	54	62	82	63	61	17

Asymetria termiczna procesora

Pomijając wpływ temperatury można stwierdzić że typowy współczesny procesor wielordzeniowy jest symetryczny pod względem wydajności. Oznacza to, że pomijając wpływ efektu zimnych pamięci podręcznych, niezależnie na który rdzeń zaszeregowany zostanie wątek, będzie on przetwarzany z taką samą prędkością. Zmierzone przez autora różnice mieszczą się w granicach niepewności pomiaru. Natomiast uwzględniając pracę przy ograniczeniu temperaturowym, mamy do czynienia z pewną asymetrią pod względem wydajności poszczególnych rdzeni.

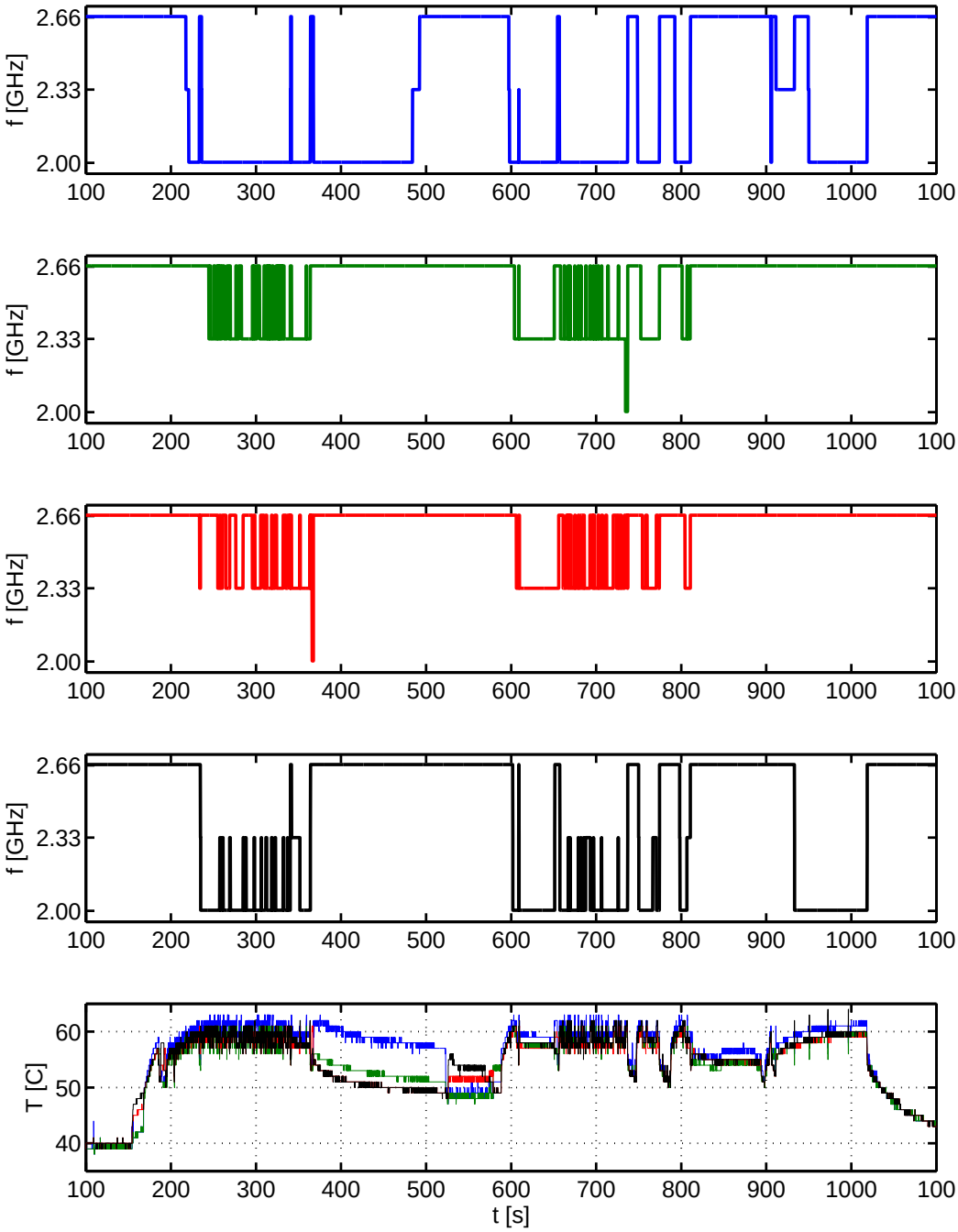
Asymetria ta objawia się poprzez nierównomierny rozkład temperatur pomiędzy poszczególnymi rdzeniami [133], a w efekcie nierównomierne obniżanie poziomów napięć i częstotliwości poszczególnych rdzeni nawet przy jednakowym obciążeniu zadaniami. Jest to spowodowane kilkoma czynnikami. Po pierwsze umiejscowienie rdzenia

w procesorze wpływa na jego możliwości oddawania ciepła do otoczenia. Rdzenie umieszczone na brzegach układu mają inną rezystancję termiczną w kierunku poziomym, niż te posiadające sąsiadów po obu stronach. Dodatkowo, wariancja procesowa może powodować zawyżone lub zaniżone odczyty temperatury oraz wpływać na poziom mocy rozpraszanej przez poszczególne rdzenie procesora.

W efekcie, mechanizmy DTM mogą powodować znaczące różnice wydajności pomiędzy poszczególnymi rdzeniami, poprzez obniżanie częstotliwości pracy rdzeni o wyższej temperaturze. Podczas przetwarzania programów wielowątkowych, wątki uruchomione na „szybszych" rdzeniach muszą czekać na synchronizację z wątkami uruchomionymi na „wolniejszych" rdzeniach. Jednocześnie rdzenie pracujące z wyższą częstotliwością powodują nadmierne zużycie energii i podwyższają temperaturę całego układu.

Aby ocenić wpływ zjawiska asymetrii wydajności spowodowanej przez mechanizm DTM przeprowadzono eksperyment polegający na uruchomieniu kilku programów z zestawu `PARSEC` podczas działania prostego algorytmu DVFS. Zarejestrowane przebiegi temperatury oraz napięć i częstotliwości przedstawiono na rysunku 5.1. Jak można zauważyć, pierwszy i ostatni rdzeń procesora dużo częściej przełączany był na minimalny poziom częstotliwości, niż dwa rdzenie umieszczone pomiędzy nimi (schemat procesora na rys. 3.4). Podczas wykonywania zadań testowych, średnia częstotliwość poszczególnych rdzeni wyniosła odpowiednio 2.27, 2.58, 2.60 i 2.40 GHz, co przekłada się na różnice w wydajności poszczególnych rdzeni na poziomie bliskim 15%. Żeby wykluczyć różnice pomiędzy poszczególnymi wątkami, powtórzono eksperyment uruchamiając cztery instancje programu `cpuburn` stosując ten sam algorytm ograniczający temperaturę. Uzyskane wyniki potwierdziły poprzednie spostrzeżenia.

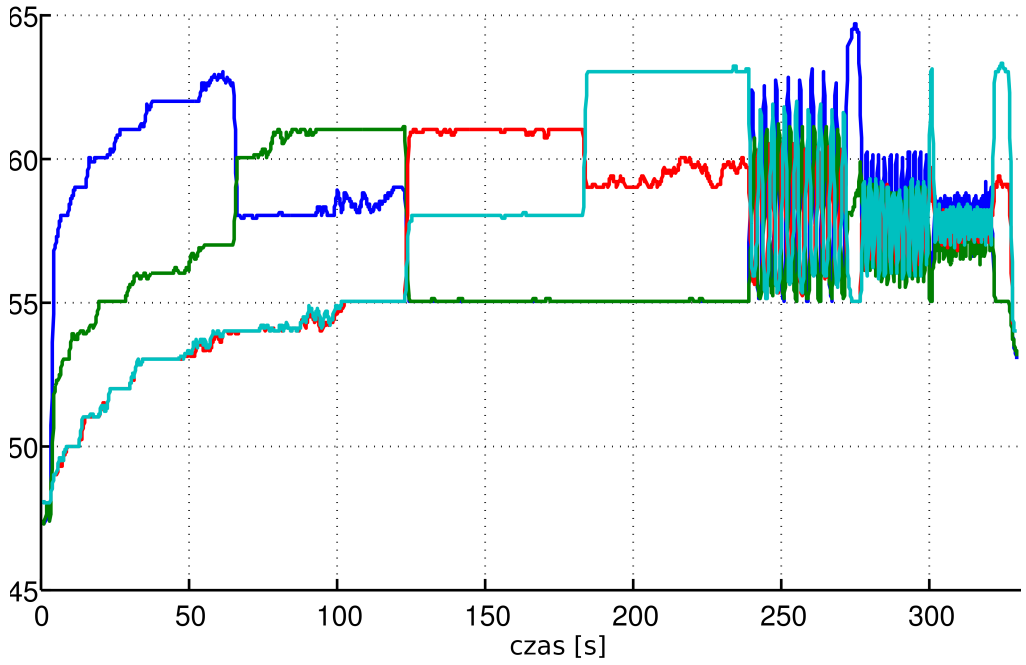
Bezpośrednim wnioskiem wynikającym z przeprowadzonego eksperymentu jest konieczność całościowego sterowania wydajnością procesora, dodatkowo uwzględniając właściwości uruchomionych na nim zadań. Algorytm który dobiera częstotliwość każdego z rdzeni osobno, opierając się jedynie na aktualnym poziomie temperatury danego rdzenia będzie z pewnością nieefektywny pod względem wydajności i zużycia energii całego procesora.



Rysunek 5.1: Przebieg częstotliwości poszczególnych rdzeni oraz ich temperatura podczas wykonywania testów z zestawu `PARSEC`

5.1.2 Migracja zadań

Oprócz skalowania napięcia i częstotliwości, rozpatrywany algorytm DTM bazuje na migracji wątków (ang. *Thread Migration – TM*). Jest to mechanizm ortogonalny



Rysunek 5.2: Profil temperatury czterech rdzeni procesora podczas migracji jednego zadania z rosnącą częstotliwością

w stosunku do DVFS. Ideą jego działania jest wyrównanie gradientów temperatury na powierzchni procesora poprzez przenoszenie wątków pomiędzy rdzeniami. Przykład skuteczności migracji przedstawiono na rysunku 5.2. Program testowy uruchomiony na pierwszym rdzeniu procesora spowodował wystąpienie różnicy temperatur na poziomie 10°C. Następnie zastosowanie migracji zadania pomiędzy rdzeniami procesora ze zwiększającą się częstotliwością spowodowało zmniejszenie gradientów i maksymalnej temperatury. Należy zaznaczyć, że migracja nie powoduje obniżenia średniej temperatury procesora, ponieważ nie obniża mocy dynamicznej.

Wykładnicza zależność prądu upływu od temperatury oraz prognozowany wzrost proporcji mocy statycznej do dynamicznej w kolejnych generacjach procesorów uzasadniają obniżanie maksymalnej temperatury w procesorze [8]. Spowoduje to obniżenie poziomu prądu upływu oraz mocy statycznej rozpraszanej przez procesor. Podstawową przesłanką do migracji wątku jest wysoka różnica temperatur pomiędzy rdzeniami procesora. Taka sytuacja występuje zazwyczaj kiedy mała liczba, względnie „gorących” wątków jest wykonywana i powoduje wzrost temperatury powyżej dopuszczalnego poziomu. W takiej sytuacji skalowanie napięcia i częstotliwości procesora

skutkuje obniżeniem wydajności, podczas gdy okresowa migracja wątków pozwala na pracę procesora przy maksymalnej częstotliwości.

Negatywnym skutkiem migracji wątku pomiędzy dwoma rdzeniami jest utrata wydajności związana z efektami zimnych pamięci (ang. *cold cache*). Po przeniesieniu wątku, pamięci podręczne docelowego rdzenia procesora nie zawierają kopii kodu ani danych wątku, więc następuje chwilowy wzrost liczby chybień. Skutkiem tego jest oczekiwanie na przesłanie danych z pamięci głównej i spowolnienie wykonywania zadania. Znaczenie tego efektu zależy od aktualnego rozmiaru zbioru roboczego. Opóźnienie przetwarzania związane z pojedynczą migracją waha się w granicach od $2\,\mu s$ do około $1\,ms$ [19]. Ponieważ pomiędzy poszczególnymi fazami programów występują znaczące różnice w intensywności dostępu do pamięci, migracja w trakcie faz charakteryzujących się niskim współczynnikiem chybień w pamięci podręcznej ostatniego poziomu spowoduje umiarkowane wydłużenie czasu przetwarzania.

Niestety, proste algorytmy migracji przenoszące wątki na kolejne rdzenie albo na losowo wybrany rdzeń mogą skutkować znacznym wydłużeniem czasu wykonania programów. Przy założeniu, że poszczególne wątki nie różnią się zachowaniem, efektywnym algorytmem jest okresowe migrowanie wszystkich wątków na losowo wybrane rdzenie. Tego typu proste algorytmy skutkują dobrymi wynikami w wysokopoziomowych symulacjach. W praktyce ilość energii rozpraszanej przez rdzeń procesora zmienia się znacząco w zależności od wykonywanego programu oraz jego aktualnej fazy. Wzięcie pod uwagę dynamicznego zachowania programu podczas migracji skutkuje mniejszą utratą wydajności i skuteczniejszym ograniczeniem gradientów temperatury.

Podejmowanie decyzji o migracji programu pomiędzy rdzeniami procesora jest dodatkowo utrudnione z powodu zależności pomiędzy logicznymi rdzeniami procesora. Algorytm migracji musi uwzględnić model termiczny procesora, a w najprostszym wariancie sąsiedztwo pomiędzy poszczególnymi rdzeniami[1]. Migracja wątku pomiędzy sąsiadującymi rdzeniami będzie mniej skuteczna niż pomiędzy rdzeniami bardziej oddalonymi od siebie na powierzchni procesora. Z drugiej strony, migracja zadań będzie się wiązała z relatywnie mniejszym kosztem, jeśli zostanie przeprowadzona pomiędzy rdzeniami współdzielącymi pamięć podręczną ostatniego poziomu. Niestety obie reguły stoją ze sobą w sprzeczności.

Kolejnym czynnikiem komplikującym zadanie migracji wątków jest współbieżna

[1]Podstawowe informacje o topografii procesora dostępne są w systemie `Linux` poprzez interfejs `sysfs` w katalogu `/sys/devices/system/cpu/cpuX/topology/`.

wielowątkowość (ang. *Simultaneous MultiThreading – SMT*). Współdzielenie zasobów jednego rdzenia procesora przez kilka wątków, jak to ma na przykład miejsce w procesorach z technologią `HyperThreading` [134] lub procesorach z rodziny `Ultra SPARC T` [135], dodatkowo komplikuje problem przyporządkowania wątków do rdzeni. Podobny efekt występuje w procesorach zbudowanych na mikroarchitekturze `Bulldozer` [136], w której rdzenie połączone są parami w moduły i współdzielą, oprócz pamięci podręcznych poziomu drugiego, układ logiki wydawczej (ang. *issue logic*).

W literaturze przedmiotu, najczęściej rozpatrywany jest problem migracji programów jednowątkowych. Usunięcie zależności pomiędzy przyporządkowaniem zasobów procesora, a wydajnością przetwarzania poszczególnych zadań sprowadza problem migracji wątków do oceny wpływu tego przyporządkowania na temperaturę. Problem wpływu przyporządkowania na wydajność przetwarzania komplikuje się znacząco w przypadku programów wielowątkowych. Współdzielenie zasobów i komunikacja pomiędzy wątkami sprawia, że migracja jednego z wątków może mieć negatywny wpływ na wydajność pozostałych wchodzących w skład tego samego programu. Dodatkowo, migracja programu może mieć wpływ na wydajność całego systemu, na przykład poprzez zajęcie magistrali w związku z koniecznością wypełnienia pamięci podręcznych.

Narzuty migracji wątków

Poniżej przedstawiono wyniki przeprowadzonych eksperymentów mających na celu określenie narzutów związanych z migracją wątku w procesorze wielordzeniowym. Skuteczność migracji zadań jako metody wyrównania gradientów temperatury i ograniczania temperatury maksymalnej była wielokrotnie podkreślana w literaturze i została zweryfikowana w trywialnych przypadkach. Na przykład w pracy [58] pokazano, że kosztem spowolnienia wykonania programów nie przekraczającym 2% możliwe jest ograniczenie temperatury maksymalnej nawet o 5,5°C, jeśli pojedynczy program przenoszony jest co określony czas pomiędzy dwoma dostępnymi rdzeniami.

Z zupełnie inną sytuacją mamy do czynienia w wypadku programów wielowątkowych. Dla przykładu programy testowe z zestawu `PARSEC` [112] wykazują szereg właściwości które powodują, że prosty algorytm migrowania zadań będzie nieefektywny pod względem redukcji temperatury i jednocześnie będzie powodował znaczące straty

wydajności. Duży zbiór roboczy programów i znacząca ilość komunikacji pomiędzy poszczególnymi wątkami powodują, że każdorazowa zmiana zaszeregowania wątku skutkuje koniecznością aktualizacji pamięci podręcznych i przestojami w przetwarzaniu. Ponadto niektóre programy charakteryzują się złożonym modelem zrównoleglenia z wieloma heterogenicznymi wątkami, które z różną intensywnością wykorzystują zasoby procesora.

W celu oceny skuteczności i strat wydajności spowodowanych przez migracje zaimplementowano dwa proste algorytmy migracji: *karuzelowy* oraz *różnicowy*. Migracja karuzelowa co ustalony czas przenosi aktywne zadania na sąsiedni rdzeń o wyższym indeksie. Algorytm migracji różnicowej polega na okresowym uszeregowaniu rdzeni względem ich temperatury. Jeśli różnica temperatur pomiędzy najzimniejszym i najgorętszym rdzeniem przekracza ustalony próg, zadania pomiędzy nimi są wymieniane. Następnie rozpatrywane są kolejne pary, dopóki różnica temperatur jest dostatecznie duża. Taki algorytm, choć prosty w implementacji pozwala na uniknięcie zbędnych migracji i jednoczesne wyrównanie gradientów. Wymieniając zadaniami rdzenie o największej różnicy temperatur pozwala też wykonywać migrację stosunkowo rzadko.

Tabela 5.4: Czas działania, temperatura maksymalna, liczba okresów w których temperatura przekraczała T_{max} dla wybranych programów z zestawów `SPEC CPU 2000` i `PARSEC` w zależności od algorytmu migracji wątków

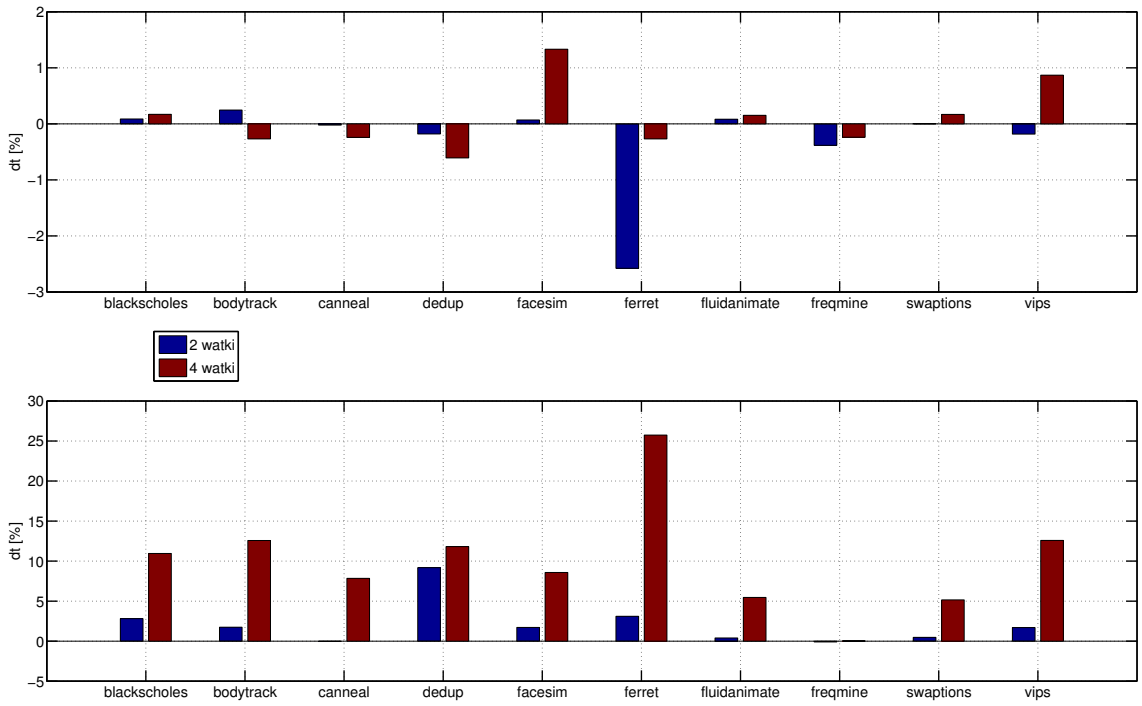
	bez migracji		m. karuzelowa			m. różnicowa		
benchmark	t [s]	T>max T	t [s]	T>max T	n_migT	t [s]	T>max T	n_migT
SPEC int	359	349	357	239	4255	356	171	40
SPEC fp	495	945	501	507	4986	504	681	36
blackscholes	93	0	104	0	3493	94	0	6
bodytrack	78	0	88	0	1364	79	0	0
canneal	130	0	139	0	1437	130	0	0
dedup	23	0	37	0	287	23	0	0
facesim	234	676	254	170	3214	237	224	3
ferret	135	174	136	130	1412	135	173	0
fluidanimate	160	589	162	489	2345	161	598	1
freqmine	730	0	779	0	4862	726	0	4
swaptions	109	333	112	292	1359	103	292	0
vips	47	54	56	47	480	51	58	0

W tabeli 5.4 przedstawiono dane uzyskane z pomiarów przeprowadzonych podczas wykonywania programów z zestawów SPEC CPU 2000 i PARSEC z zastosowaniem 2 algorytmów migracji oraz pod kontrolą standardowego mechanizmu szeregowania zadań w systemie Linux. Każdy z pomiarów przeprowadzony był dziesięciokrotnie a w tabeli umieszczono wartości środkowe. Poszczególne uruchomienia programów oddzielone były od siebie przerwami o długości 60 s w celu ochłodzenia procesora i zapewnienia porównywalnych warunków termicznych. Programy jednowątkowe, z zestawu SPEC CPU 2000 uruchamiane były kolejno w grupach po cztery. Po zakończeniu ostatniego z nich, startowała następna grupa. Programy z pakietu PARSEC uruchamiane były pojedynczo, zrównoleglone na 4 rdzenie procesora.

Na podstawie przedstawionych danych można zauważyć, że migracja karuzelowa powoduje zauważalne zwiększenie czasu przetwarzania przy jednoczesnym obniżeniu temperatury. Drugi z algorytmów migracji skutkuje minimalnym wydłużeniem czasu przetwarzania. Ponadto część programów z pakietu PARSEC nie spowodowała przekroczenia temperatury maksymalnej. Spowodowane to było albo krótkim czasem przetwarzania (program dedup) w stosunku do stałej czasowej radiatora, lub też niską intensywnością obliczeń spowodowaną przez dużą liczbę operacji wejścia-wyjścia.

Na rysunku 5.3 przedstawiono wartości środkowe wartości spowolnienia wykonywania programów spowodowanego algorytmem różnicowym i karuzelowym. Można zauważyć wyraźną przewagę pierwszego z algorytmów. Co więcej wydajność wyraźnie spada jeśli program zrównoleglony jest na 4 wątki.

Wydłużenie czasu przetwarzania spowodowane pojedynczą migracją zostało oszacowane w następujący sposób: różnica środkowych długości czasu przetwarzania z migracjami i bez została podzielona przez zarejestrowaną liczbę wykonanych migracji. Maksymalny czas opóźnienia spowodowany pojedynczą migracją w przypadku programów z zestawu PARSEC zrównoleglonych na 4 rdzenie wynosił 3,1 ms. Przy dwóch wątkach, tylko w czterech przypadkach spowolnienie w przeliczeniu na pojedynczą migrację przekroczyło 1 ms. Z przedstawionych danych można wyciągnąć następujące wnioski pomocne w budowie algorytmu DTM wykorzystującego migrację wątków. Ponieważ migracje powodują wzrost liczby chybień w pamięciach podręcznych ostatniego poziomu, nie należy migrować zadań cechujących się dużym rozmiarem zbioru roboczego i wysokim współczynnikiem chybień. Programy o niskim współczynniku chybień w pamięciach podręcznych, programy o relatywnie niewielkim zbiorze roboczym, które intensywnie wykorzystują procesor i programy jednowątkowe, które



Rysunek 5.3: Spowolnienie wykonywania programów wielowątkowych z zestawu PARSEC spowodowane migracją. Wyniki dla algorytmu migracji różnicowej (góra) oraz karuzelowej (dół).

nie wymagają komunikacji przez wspólną pamięć nie są spowalniane przez częste migracje. Stąd można uzależnić próg (graniczną różnicę temperatur) od bieżącego zachowania zadania. Można również zauważyć, że migracje są najmniej efektywne gdy wszystkie rdzenie są zajęte.

5.2 Proponowany mechanizm

W poprzednich rozdziałach przedstawiono ocenę skuteczności dynamicznego skalowania napięcia i częstotliwości oraz migracji wątków jako mechanizmów zarządzania temperaturą w procesorze wielordzeniowym. Poniżej przedstawiono proponowany heurystyczny algorytm sterowania pracą procesora maksymalizujący całkowitą wydajność przy zadanym ograniczeniu na temperaturę maksymalną. Ocena skuteczności tego algorytmu przeprowadzona została na komputerze z procesorem, którego podstawowe parametry podane zostały w podrozdziale 3.1.

Proponowany mechanizm został zaimplementowany w postaci aplikacji w języ-

ku `Python`, działającej niezależnie od systemowego mechanizmu szeregowania zadań. Wadami takiego rozwiązania są: mniejsza wydajność niż jest możliwa do osiągnięcia poprzez modyfikację jądra systemu operacyjnego oraz większe narzuty związane z odczytaniem stanu procesora i informacji o uruchomionych zadaniach. Z kolei do zalet takiego podejścia należą uproszczona implementacja i łatwość modyfikacji działania takiego programu bez konieczności ponownej kompilacji jądra systemu operacyjnego. Ponadto rozdzielenie warstw upraszcza działanie całego systemu. Domyślny mechanizm szeregowania zadań w systemie `Linux` – `CFS` (ang. *Completely Fair Scheduler*) maksymalizuje wydajność jednocześnie uwzględniając priorytety programów i zapewniając sprawiedliwy przydział czasu procesora wszystkim aktywnym procesom [126].

Podstawowe reguły na jakich oparta jest proponowana metoda to:

- uwzględnienie priorytetów wykonywanych zadań,
- uwzględnienie zachowania programów – obniżenie w pierwszej kolejności częstotliwości pracy wątków, których wydajność ograniczona jest przez odwołania do pamięci głównej,
- prognoza temperatury procesora na podstawie znajomości zachowania programów i modelu termicznego procesora,
- wykorzystanie migracji jeśli różnice temperatur pomiędzy rdzeniami są większe od ustalonej wartości i jeśli liczba chybień w pamięci podręcznej ostatniego poziomu nie przekracza ustalonej wartości,
- wykorzystanie pojemności cieplnej radiatora – dopóki temperatura jest znacząco niższa niż $T_{\max}$ należy pracować z maksymalną częstotliwością.

Na podstawie prognozy temperatury można oszacować maksymalną częstotliwość pracy każdego rdzenia, która nie spowoduje przekroczenia temperatury maksymalnej. Lewa strona równania (4.4) odpowiada temperaturze prognozowanej w następnym kroku algorytmu. Jeśli utworzymy wektor kolumnowy o długości N_{RC} zawierający wartość $\mathrm{T}_{\max}$ w każdym elemencie i oznaczymy go przez $\mathbf{T}_{\max}$ to możemy zapisać:

$$\mathbf{T}_{\max} = \mathbf{ET} + \mathbf{FP}_{\max}. \tag{5.1}$$

Stąd warunek na maksymalną moc rozpraszaną przez poszczególne elementy procesora $\mathbf{P}'$ ma postać:

$$\mathbf{F}^{-1}\left(\mathbf{T}_{\max} - \mathbf{ET}\right) = \mathbf{P}_{\max} \geqslant \mathbf{P}'. \tag{5.2}$$

Następnie, korzystając z zależności (3.14) i (3.15) można znaleźć taki zestaw ustawień napięcia i częstotliwości **S**, że moc rozpraszana przez każdy z rdzeni będzie mniejsza niż moc maksymalna. Ponieważ liczba dostępnych poziomów szybkości rdzeni $s = (f, V)$ jest zwykle nie większa niż 10, wystarczy obliczać moc dla kolejnych malejących poziomów aż do znalezienia pierwszego, który nie spowoduje przekroczenia temperatury maksymalnej.

Uwzględnienie priorytetów zadań w algorytmie sterowania częstotliwością rdzeni można uzyskać poprzez przypisanie każdemu z nich wagi zależnej od priorytetu zadań na nim wykonywanych, a następnie przydzieleniu rdzeniom o wyższej wadze pierwszeństwa do zwiększenia częstotliwości procesora. W przypadku dwóch rdzeni i oraz j o różnych priorytetach $p_i < p_j$, jeśli rdzeń o wyższym priorytecie[2] p_j nie pracuje z maksymalną dostępną częstotliwością (jego wydajność jest ograniczona z powodu wysokiej temperatury), częstotliwość pracy rdzenia i powinna zostać zmniejszona. Dzięki temu obniży się całkowita energia rozpraszana przez procesor co pozytywnie wpłynie na wydajność rdzenia j. Należy zaznaczyć, że uwzględnienie priorytetów zadań może spowodować zmniejszenie sumarycznej wydajności procesora.

Zmiana częstotliwości rdzenia ma relatywnie mniejszy wpływ na czas wykonania zadań charakteryzujących się większą liczbą odwołań do pamięci głównej. Dlatego celowym jest uwzględnienie liczby chybień w pamięci podręcznej ostatniego poziomu (LLCM) w algorytmie dynamicznego sterowania napięciem i częstotliwością rdzeni. W tym celu można rdzeniowi nadać wagę zależną zarówno od jego priorytetu jak i własności przetwarzanych na nim zadań. Wagę i-tego rdzenia można obliczyć według zależności:

$$w_i = \frac{k_p - p_i}{-k_l \cdot log_{10} LLCM_i}, \tag{5.3}$$

gdzie k_p i k_l są eksperymentalnie dobranymi współczynnikami zapewniającymi sprawne działanie systemu. W systemie `Linux` priorytety procesów mają wartości zawierające się w przedziale $\langle 20, 19 \rangle$. Współczynnik k_p pozwala zmieniać wpływ priorytetu zadania na wagę. Ponieważ testowane programy różniły się w wartościach współczyn-

[2]Za priorytet rdzenia można przyjąć priorytet zadania, które jest w danym momencie na nim przetwarzane. W przypadku zaszeregowania więcej niż jednego aktywnego zadania do jednego rdzenia, można obliczyć wypadkowy priorytet będący średnią ważoną priorytetów zadań, przy czym wagą danego priorytetu jest procentowy udział tego zadania w czasie procesora.

nika LLCM[3] nawet o trzy rzędy wielkości, wyliczana waga zawiera jego logarytm. Parametry k_p i k_l o wartościach odpowiednio 30 i 2 pozwalają uwzględnić zarówno priorytet jak i zachowanie procesu. Dokładne zbadanie wpływu obu parametrów na wydajność systemu operacyjnego wymaga wielu eksperymentów, a zarazem nie wpływa na budowę proponowanego algorytmu.

5.2.1 Algorytm

Opracowana na podstawie omówionych eksperymentów i wniosków z nich płynących metoda zarządzania pracą procesora przy ograniczeniu temperaturowym (DTM) jest przedstawiona poniżej w postaci pseudokodu w algorytmie 4.

Algorytm 4 Proponowana metoda DTM

Wejście: poprzednia prognoza temperatury $\mathbf{T}$ i jej czas t_p

Wyjście: temperatura prognozowana $\mathbf{T}_p$

1: zaktualizuj współczynniki IPC i LLCM wszystkich rdzeni
2: odczytaj bieżącą częstotliwość poszczególnych rdzeni
3: odczytaj listę przetwarzanych zadań $\mathcal{Z}$
4: prognozuj temperaturę procesora $\mathbf{T}_p$ wg algorytmu 1
5: sortuj rdzenie względem malejącej prognozowanej temperatury
6: **dla kolejnych** par rdzeni $i \in \langle 0..\frac{N_c}{2}-1\rangle$ oraz $j \in \langle N_c-1..\frac{N_c}{2}\rangle$: $T_j - T_i \geqslant T_{th}^{mig}$ **wykonaj**
7: **jeśli** $LLCM_i < LLCM_{th}$ **oraz** $LLCM_j < LLCM_{th}$ **wtedy**
8: zamień zadania pomiędzy rdzeniami i oraz j
9: zaktualizuj prognozę mocy $\mathbf{P}'$ uwzględniając nowy przydział zadań
10: wyznacz maksymalną moc wszystkich rdzeni $\mathbf{P}_{\max}$ wg nierówności (5.2)
11: wyznacz wagi w_i wszystkich rdzeni na podstawie priorytetów zadań na nich uruchomionych i LLCM rdzeni wg wzoru 5.3
12: sortuj rdzenie według malejących wag
13: ustaw $f_{i-1}^{-1} \leftarrow f_{\max}$, $f_{i-1}^{0} \leftarrow f_{\max}$

[3]Przykładowe zaobserwowane w eksperymentach wartości umieszczono w tabeli 3.3.

Algorytm 4 Proponowana metoda DTM – c.d.

14: **dla kolejnych** rdzeni i posortowanych wg wag w_i **wykonaj**
15: wyznacz maksymalną częstotliwość i-tego rdzenia $f_{max,i}$ na podstawie (3.14) i (3.15) taką, że $P(f_{\max}, i) \leqslant P_{\max,i}$
16: **jeśli** częstotliwość poprzedniego rdzenia $f_{i-1}^{-1} = f_{i-1}^{0} = f_{\max}$ **wtedy**
17: ustaw $f_i \leftarrow f_{max,i}$
18: **w przeciwnym razie jeśli** częstotliwość poprzedniego rdzenia $f_{i-1}^{0} = f_{\max}$ **wtedy**
19: ustaw $f_i \leftarrow min(f_{max,i}, f_i^0)$
20: **w przeciwnym razie**
21: znajdź niższy od bieżącego poziom częstotliwości $f = max(f \in \Phi, f < f_i^{-1})$
22: ustaw $f_i \leftarrow min(f, f_{\max,i})$
23: zaktualizuj prognozę temperatury (4.4)
24: zapisz prognozowaną temperaturę $\mathbf{T}_p \leftarrow \mathbf{T}_0$

W pierwszej części algorytmu uaktualniana jest informacja o stanie procesora. Odczytywane są liczby instrukcji wykonanych w każdym rdzeniu procesora, liczby chybień w pamięciach podręcznych ostatniego poziomu i bieżąca temperatura każdego rdzenia. Na podstawie poprzedniej i nowej liczby cykli zegara obliczane są aktualne współczynniki IPC oraz LLCM poszczególnych rdzeni. Ponieważ liczniki wydajności mają w architekturze `IA-32` długość 40 bitów, liczbę instrukcji wykonanych w ostatnim okresie należy obliczać modulo 2^{40}. Odczytywane są również bieżące ustawienia częstotliwości taktowania zegara poszczególnych rdzeni. Na podstawie różnic pomiędzy prognozowaną a odczytaną temperaturą każdego rdzenia obliczane są błędy prognozy ε.

Odczytywana jest również lista aktualnie wykonywanych zadań które zajmują więcej niż L procent czasu procesora. Próg obciążenia procesora L na poziomie 10% pozwala na pominięcie programów takich jak na przykład demony systemowe, które są aktywne regularnie, ale przez krótki czas i nie mają zauważalnego wpływu na temperaturę procesora. Do każdego rdzenia przypisywany jest priorytet jaki mają w systemie operacyjnym procesy na nim wykonywane. W większości przypadków liczba aktywnych zadań jest mniejsza niż liczba dostępnych rdzeni procesora, co jest celowe z uwagi na wydajność przetwarzania. W przypadku gdy do jednego rdzenia zaszeregowany jest więcej niż jeden program, rdzeniowi przypisywany jest najwyższy

z ich priorytetów. Przypisanie priorytetów proporcjonalnie do czasu zajętości rdzenia da podobny wynik, ponieważ systemowy mechanizm szeregowania zadań przydziela więcej czasu procesora zadaniom o wyższym priorytecie.

W kolejnym kroku algorytmu obliczana jest prognoza temperatury poszczególnych rdzeni na podstawie algorytmu 1. Następnie, rdzenie sortowane są pod względem temperatury w porządku malejącym. Zaczynając od rdzeni o najwyższej i najniższej temperaturze analizowane są kolejne pary. Jeśli różnica temperatury przekracza wartość progową T_{th}^{mig} i liczba chybień w pamięci podręcznej obu rdzeni nie przekroczyła ustalonej wartości, zadania uruchomione na rdzeniu „gorącym" przenoszone są na rdzeń „zimny". Jeśli do chłodniejszego z rdzeni przypisane są aktywne zadania, są one przenoszone na rdzeń o wyższej temperaturze. W razie gdy do jednego z rdzeni nie było przypisanych zadań a do drugiego był przypisany więcej niż jeden proces, część z nich przenoszona jest na rdzeń chłodniejszy. Ponieważ brakuje informacji o zachowaniu obydwu procesów, na chłodniejszy rdzeń przenoszony jest proces, lub procesy o sumarycznym czasie zajętości procesora większym niż 50%.

Migracja pozwala zredukować różnice temperatury na powierzchni procesora, ale jednocześnie zmienia prognozowaną ilość energii rozproszoną przez poszczególne rdzenie. Z tego powodu należy przeliczyć prognozę temperatury ponownie, powtarzając kroki 4, 5 i 12 algorytmu 1 dla każdego z rdzeni którego dotyczyła migracja i obliczając prognozowaną temperaturę wg wzoru (4.4). Ponadto przenosząc zadanie na inny rdzeń należy skopiować zawartość tablic prognozy współczynników IPC i LLCM.

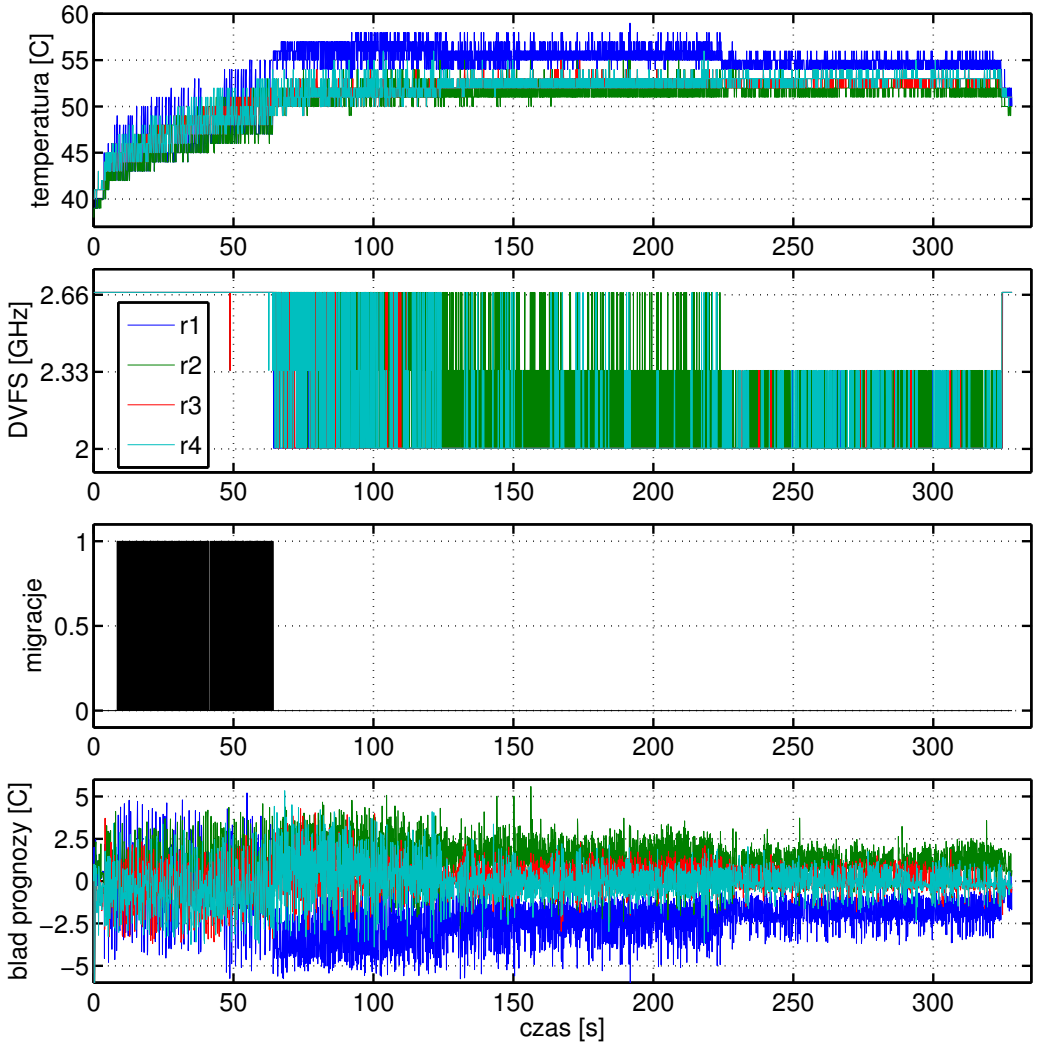
W kolejnym etapie obliczana jest maksymalna częstotliwość każdego z rdzeni która nie spowoduje przekroczenia $T_{\max}$ po czasie t_s. Na podstawie równania (5.2) obliczana jest maksymalna dozwolona moc w każdym rdzeniu. Aby obliczyć pożądaną częstotliwość należy posłużyć się zależnościami (3.14) oraz (3.15), które pozwalają oszacować moc dynamiczną rozpraszaną przez rdzeń na podstawie wartości dotychczasowej i zadanej częstotliwości taktowania. Uwzględnienie priorytetów wykonywanych zadań wymaga przydzielenia wyższych częstotliwości taktowania rdzeniom przetwarzającym programy o wyższym priorytecie, nawet kosztem wydajności przetwarzania zadań z niższym priorytetem. Ponadto, wykonanie programów, które wymagają dużej liczby odwołań do pamięci głównej, nie jest spowalniane proporcjonalnie do częstotliwości procesora. Poszczególnym rdzeniom przypisywane są wagi określające kolejność, według której dobierane są ich częstotliwości pracy. Wagi obliczane są zgodnie z zależnością (5.3) i odzwierciedlają priorytety zadań uruchomionych na kolejnych rdzeniach

oraz spodziewany wpływ skalowania częstotliwości zegara na wydajność przetwarzania tych zadań.

Rdzeniowi o największej wadze przypisuje się maksymalną częstotliwość, która nie spowoduje przekroczenia mocy maksymalnej obliczonej na podstawie nierówności (5.2). Kolejnym według malejących wag rdzeniom przypisuje się maksymalny dopuszczalny poziom częstotliwości według następujących reguł. Jeśli poprzedni rdzeń j o wyższej lub równej wadze ($w_j \geqslant w_i$) działał w poprzednim kroku z maksymalną częstotliwością $f_{\max}$ i przypisano mu maksymalną częstotliwość na nowy okres, częstotliwość rdzenia i ustalana jest tylko na podstawie wcześniej obliczonego limitu mocy rozpraszanej. Jeśli rdzeń j o wyższej wadze działał w poprzednim kroku z częstotliwością mniejszą niż $f_{\max}$, rdzeniowi i nie należy zwiększyć częstotliwości pracy i przydzielona mu zostaje mniejsza z dwóch częstotliwości – dotychczasowej i obliczonej na podstawie limitu mocy. Natomiast, jeśli rdzeniowi o wyższej wadze przypisano częstotliwość mniejszą od maksymalnej, częstotliwość kolejnego powinna zostać zmniejszona, tak aby w następnym kroku temperatura całego procesora została obniżona i możliwe było zwiększenie częstotliwości rdzenia wykonującego proces o wysokim priorytecie. Rdzeniowi o niższej wadze przypisuje się mniejszy z dwóch poziomów częstotliwości: dotychczasowego zmniejszonego o jeden i wyliczonego na podstawie limitu mocy.

W końcowym etapie ponownie przeliczana jest moc dynamiczna z uwzględnieniem nowego przydziału zadań do rdzeni oraz zmienionych częstotliwości taktowania. Uaktualniony wektor prognozy $\mathbf{T}_p$ oraz czas t_p obliczenia prognozy (określony cyklem procesora) zapisywane są do kolejnego wywołania algorytmu sterowania pracą procesora.

Na rysunku 5.4 przedstawiono dane zebrane podczas eksperymentu z zastosowaniem zaproponowanego mechanizmu DTM i programu `burnP6`. Po 4 sekundach od początku eksperymentu uruchomiona została jedna instancja programu, który wyraźnie podnosił temperaturę procesora. Aby wyrównać temperaturę, algorytm przenosił to zadanie między rdzeniami. W ciągu 1 minuty, algorytm wykonał 192 migracje co daje średnią 289 ms na migrację – taka częstotliwość wystarczyła aby zapewnić małe (ok. 5°C) różnice temperatur między rdzeniami. Pomiędzy 64, a 124 sekundą uruchomione były dwie instancje programu `burnP6`. W tym przypadku migracja nie wystarczyła do obniżenia temperatury do oczekiwanego poziomu i algorytm dobierał częstotliwości pracy poszczególnych rdzeni które zapewniały temperaturę niższą od



Rysunek 5.4: Przykład działania algorytmu: temperatura i częstotliwość zegara rdzeni procesora, migracje zadań i błędy prognozy temperatury.

ustalonego progu (55°C). W 124 i 224 sekundzie uruchomione zostały trzecia i czwarta instancja programu. Można zauważyć, że temperatura zatrzymuje się na poziomie niewiele wyższym niż oczekiwane 55°C. Odpowiada za to migracja w pierwszym okresie (wykres trzeci od góry), a następnie skalowanie częstotliwości (wykres drugi). Relatywnie duży błąd prognozy wynika z dużego okresu prognozy wynoszącego 100 ms.

Na drugim wykresie na rysunku 5.4 widać, że algorytm wykonuje bardzo wiele zmian poziomu napięcia i częstotliwości aby ograniczyć temperaturę, a przy tym dobrać maksymalny poziom wydajności. Jedna zmiana poziomu napięcia wiąże się ze wstrzymaniem wykonania zadania na 10 μs, a więc nawet gdyby zmiany wykonywa-

ne były przy każdym wywołaniu algorytmu, skutkowałoby to spowolnieniem pracy o mniej niż jeden promil. Ponadto średnie uzyskane częstotliwości (w GHz) w kolejnych etapach testu wyniosły [2.66, 2.66, 2.66, 2.66], [2.02, 2.46, 2.09, 2.23], [2.00, 2.20, 2.00, 2.04] i [2.02, 2.06, 2.03, 2.06]. Były to wartości odległe od dostępnych częstotliwości procesora. Dlatego, aby zmaksymalizować wydajność konieczne są częste zmiany częstotliwości pracy. Ponadto, przekształcając równanie 3.13 do postaci 5.4:

$$s(f) = f\left(\frac{s(f_{\max})}{f_{\max}} - \frac{LLCM \cdot k_E}{f_{\max}}\right) + LLCM \cdot k_E, \tag{5.4}$$

można stwierdzić, że przyjęty model wydajności jest liniowy względem częstotliwości f, a więc częste zmiany częstotliwości nie wpływają znacząco negatywnie na wydajność.

Warto jeszcze zwrócić uwagę na to jak błąd prognozy wpływa na skuteczność ograniczania temperatury. Pomiędzy 64 a 224 sekundą prognoza temperatury rdzenia ***r1*** była zaniżana[4], co przekłada się na zauważalne przekroczenie temperatury maksymalnej. W kolejnych wywołaniach algorytmu prognozowana temperatura jest niższa od rzeczywistej, więc algorytm dobiera wysoki poziom napięcia. W efekcie ustalony poziom temperatury jest często przekraczany.

5.3 Wyniki eksperymentów

Testowa implementacja prezentowanego algorytmu sterowania pracą procesora pod ograniczeniem na maksymalną temperaturę została stworzona w języku `Python`. Do sterowania częstotliwością pracy procesora i przydziałem rdzeni do procesów wykorzystano standardowe mechanizmy systemu `Linux`: moduł jądra systemu operacyjnego `cpufreq` i program `taskset`. Lista aktywnych procesów pobierana jest przy każdym wywołaniu algorytmu programem `ps`. Informacje o temperaturze i bieżącej aktywności procesora odczytywane są z rejestrów MSR poszczególnych rdzeni. Zaimplementowany algorytm został przetestowany na komputerze z czterordzeniowym procesorem `Intel Core 2 Quad Q9400`, którego parametry podano w rozdziale 3.1.

Ponieważ program sterujący musi konkurować o zasoby procesora z programami testowymi, może się zdarzyć, że kolejne wywołania algorytmu nie będą występowały po ustalonym okresie. Może to zwiększyć błąd prognozy oraz opóźnić reakcję na

[4]Powód występowania tego błędu opisano w rozdziale 3.4.

przegrzewanie się procesora. W celu zapewnienia jednakowych odstępów między kolejnymi wywołaniami algorytmu wykorzystano mechanizm timerów oraz zwiększono priorytet procesu sterującego względem programów testowych.

Do oceny skuteczności prezentowanego algorytmu DTM przygotowano skrypty, które uruchamiały program sterujący wydajnością procesora i programy testowe. Weryfikacja skuteczności algorytmu dla programów jednowątkowych przeprowadzona została przy użyciu zestawu `SPEC CPU 2000`. Programy jednowątkowe uruchamiane były grupami po cztery. Po zakończeniu wszystkich programów, uruchamiana była kolejna grupa, aż do zakończenia całego pakietu. Ponieważ czas działania programów był różny, występowały okresy różnej zajętości procesora – od jednego do 4 aktywnych zadań. Odpowiada to zmiennemu obciążeniu w rzeczywistych zastosowaniach. Do oceny działania algorytmu dla programów wielowątkowych wykorzystano programy z zestawu `PARSEC` z domyślnymi danymi wejściowymi. Testy przeprowadzane były z liczbą wątków wynoszącą od dwóch do czterech. Do przeprowadzenia eksperymentów przygotowano skrypty powłoki `bash`, które uruchamiały testy z zadanymi parametrami. Każdy kolejny zestaw testów włączany był po ustabilizowaniu temperatury komputera przez 10 minut. Aby zapewnić jednakowe warunki we wszystkich eksperymentach, temperatura otoczenia procesora T_{amb} była monitorowana. Jeżeli w momencie rozpoczęcia testu T_{amb} różniła się od 28°C o więcej niż 1°C, test był unieważniany i powtarzany.

Wszystkie testy powtarzane były dziesięciokrotnie aby zminimalizować możliwość wpływu losowych zdarzeń na czas wykonywania programów. Po odrzuceniu wyników z najdłuższym i najkrótszym czasem wykonania, z pozostałych wartości obliczana była średnia. Podawana temperatura maksymalna osiągnięta dla danego zestawu parametrów jest maksymalną temperaturą osiągniętą we wszystkich próbach.

Wyniki eksperymentu przedstawione zostały w tabeli 5.5. Zawiera ona czas wykonania testów, maksymalną osiągniętą temperaturę i jej średnią wariancję. Skuteczność poszczególnych metod DTM oceniana jest na podstawie sumarycznej wydajność procesora przy ograniczeniu maksymalnej temperatury pracy. Przekłada się to na całkowity czas wykonania ustalonych zadań. Kolejnym kryterium jest osiągnięta maksymalna temperatura oraz liczba okresów, w których poziom $T_{\max}$ był przekroczony. Z uwagi na konieczność maksymalizacji wydajności praca w temperaturze bliskiej $T_{\max}$ jest pożądana, co umożliwia sporadyczne przekroczenie wartości granicznej z uwagi na nieprzewidziane sytuacje. Może to być na przykład uruchomienie nowego procesu

Tabela 5.5: Wyniki uzyskane dla proponowanego algorytmu i różnych metod DTM

	SPEC int	SPEC fp	PARSEC N=2	PARSEC N=3	PARSEC N=4
	czas wykonania [s]				
(pred)-none	453	655	3175	2501	2633
dvfs	502	745	3380	2752	2949
pred-dvfs	495	732	3431	2647	2891
tm	456	645	3511	2929	3096
pred-tm	456	650	3317	2700	2637
prez. alg.	472	732	3360	2695	2811
	maksymalna temperatura [°C]				
pred-none	68	68	69	67	70
dvfs	59	60	60	59	59
pred-dvfs	58	59	59	58	58
tm	67	69	64	67	70
pred-tm	69	68	67	67	70
prez. alg.	58	57	57	56	58
	liczba okresów t: $max(T) > T_{\max}$				
pred-none	4359	6746	21780	14704	20919
dvfs	52	30	8937	100	3503
pred-dvfs	17	3	15	1	13
tm	4214	6704	23794	14872	23009
pred-tm	4498	6647	19978	14079	21985
prez. alg.	0	0	5	0	11
	średnia wariancja temperatury rdzeni				
pred-none	2.82	1.68	5.37	4.60	4.74
dvfs	1.49	0.63	3.39	2.85	3.25
pred-dvfs	1.66	0.69	3.92	3.66	3.69
tm	2.15	1.28	2.99	2.59	2.29
pred-tm	4.27	1.68	3.69	3.00	3.04
prez. alg.	1.23	0.61	1.83	1.98	1.81

lub opóźnienie wywołania algorytmu sterowania. Uwzględniając występujący szum i rozdzielczość czujników temperatury wynoszącą 1°C, zliczano jedynie przekrocze-

nia $T_{\max}$ o 2°C i więcej. Następną miarą skuteczności algorytmu DTM jest wariancja temperatury między rdzeniami procesora. Ponieważ może przyjmować ona różne wartości, zależnie od chwilowego obciążenia rdzeni zadaniami, w tabeli 5.5 podano średnią wariancję w każdym z eksperymentów.

Na podstawie wyników przedstawionych w tabeli 5.5 można zauważyć, że poszczególne metody DTM mają różny wpływ na czas wykonywania programów jedno i wielowątkowych. Zastosowanie DVFS wydłuża czas przetwarzania w każdym przypadku, zapewniając przy tym dość dobrą skuteczność w ograniczaniu temperatury rdzeni procesora. Natomiast migracja zadań w przypadku programów wielowątkowych wiąże się z dużą utratą wydajności. Ten efekt ma minimalne znaczenie w przypadku programów jednowątkowych z pakietu `SPEC`.

Można również zauważyć, że przy przy ograniczeniu temperaturowym dodawanie kolejnych wątków nie przyspiesza znacząco wykonania programów z pakietu `PARSEC`. Uruchomienie dodatkowych zasobów procesora powoduje dalszy wzrost temperatury i w efekcie konieczność zmniejszenia częstotliwości pracy rdzeni.

Nie należy porównywać wyników pomiędzy różnymi testami, ponieważ mają one różną intensywność i czas wykonywania.

5.4 Podsumowanie

Migracja jest najbardziej skuteczna kiedy wątków jest znacząco mniej niż rdzeni ($N_{th} < \frac{1}{2}N_c$). Ponadto, zaobserwowano wysokie różnice temperatur między rdzeniami bezpośrednio po zakończeniu jednego z zadań na mocno obciążonym procesorze. W wysoce obciążonym systemie różnica temperatur między radiatorem a rdzeniem procesora jest niższa niż w mało obciążonym systemie. Ilość ciepła przekazywana przez powierzchnię między procesorem a radiatorem wielokrotnie przekracza ilość ciepła wymienianą bezpośrednio między rdzeniami. Wynika stąd, że temperatura rdzenia jest w głównej mierze determinowana przez aktualnie rozpraszaną w nim moc oraz różnicę temperatur między tym rdzeniem, a częścią radiatora bezpośrednio nad nim.

Metoda prognozy temperatury i sterowania wydajnością procesora musi być efektywna obliczeniowo. Dla przykładu metoda wyboru częstotliwości pracy procesora, zaimplementowana w środowisku MATLAB, wg autorów pracy [44] wymaga 0,5 do 1,5 s czasu na obliczenia. Jej autorzy szacują, że implementacja w języku C może być od 10 do 100 razy wydajniejsza. Nawet najbardziej optymistyczny wariant, w którym

wybór ustawień DVFS wymaga 5 ms obliczeń jest, z uwagi na ograniczenia prezentowane w tym oraz poprzednim rozdziale, nie do przyjęcia w praktycznym zastosowaniu. Ponadto, rozproszona metoda [44] może prowadzić do całkowitego wstrzymywania pracy rdzenia, gdy zakres dostępnych ustawień DVFS nie wystarczy do ograniczenia temperatury pracy poniżej ustalonego limitu.

W większości prac, w których rozważa się wykorzystanie migracji wątków do wyrównywania temperatur między rdzeniami, wykorzystuje się metody analityczne i symulacyjne, przyjmując przy tym bardzo uproszczony model zadania (np. w pracy [55]). W pracach opartych na pomiarach wydajności i temperatury rzeczywistych procesorów wykorzystuje się zwykle programy jednowątkowe[5]. W świetle wyników przedstawionych w powyższym rozdziale można stwierdzić, że migracja zadań może mieć duży wpływ na wydajność jeśli dotyczy programów wielowątkowych. Wobec upowszechniania się procesorów wielordzeniowych i programów wielowątkowych potrzebna będzie weryfikacja dotychczasowej wiedzy w tym temacie i ostrożne podejście do migracji zadań oraz szeregowania z uwzględnieniem temperatury. W przypadku gdy wykonywanych jest niewiele zadań jednowątkowych o relatywnie małym rozmiarze zbioru roboczego, migracja wątków stanowi skuteczną metodę redukcji temperatury.

Ważnym parametrem wpływającym na skuteczność mechanizmu sterowania wydajnością procesora jest częstotliwość aktualizacji prognozy. Prognoza obliczana zbyt rzadko może powodować skutek odwrotny do zamierzonego – powodować wpadanie algorytmu w oscylacje. Pomiędzy kolejnymi wywołaniami algorytmu temperatura rdzenia może zmieniać się tak bardzo, że jego szybkość będzie zmieniana w każdym kroku między dwoma skrajnymi ustawieniami. Powoduje to niepotrzebne zmiany poziomu napięcia rdzenia, skutkujące wstrzymywaniem wykonania programu oraz pracę w temperaturze przekraczającej wyznaczony limit.

Podczas wykonywania eksperymentów na rzeczywistych maszynach należy zwracać uwagę na warunki otoczenia, w szczególności temperaturę pomieszczenia, w którym znajduje się komputer. Zmiana temperatury nawet o 2–3 °C może wpływać na ilościową ocenę skuteczności algorytmu – zwiększając częstość przekroczeń limitu temperatury lub zmniejszając wydajność. Jeśli porównuje się dwa lub więcej algorytmów, może się okazać, że taka zmiana temperatury otoczenia spowoduje odwrócenie

[5]Najczęściej wykorzystywane są wybrane programy z pakietów `SPEC CPU 2000` i `2006` (np. [57, 86]), jednak należy zaznaczyć, że wybranie podzbioru programów może prowadzić do błędnych wniosków [137].

rzeczywistej kolejności. Podobnie należy mieć na uwadze, aby kolejne eksperymenty nie wpływały na siebie. Jeśli nie zachowa się dostatecznie dużej przerwy pomiędzy kolejnymi testami, to ich kolejność może mieć wpływ na uzyskane wyniki.

Rozdział 6

Wnioski i dalsze kierunki badań

W niniejszej rozprawie poruszono problemy związane z temperaturą procesorów wielordzeniowych. Temperatura od przeszło dekady stanowi jeden z najważniejszych czynników limitujących wydajność procesorów. Rosnąca w kolejnych generacjach procesorów gęstość mocy oraz ograniczenia w możliwości odprowadzania ciepła powodują niemożność jednoczesnego wykorzystania w pełni wszystkich zasobów procesorów. Jednym z głównych sposobów ograniczania emisji mocy jest sterowanie temperaturą rdzeni procesora. Zapewnienie wysokiej wydajności przy ustalonym poziomie niezawodności i zużyciu energii wymaga wiedzy na temat architektury komputerów, technologii i źródeł ciepła w procesorach, oceny wydajności programów i zarządzania zasobami procesora przez system operacyjny.

W rozprawie przedstawiono mechanizmy odpowiedzialne za powstawanie i transfer ciepła w procesorach, zależne od temperatury mechanizmy które mają wpływ na niezawodność procesorów, oraz metody modelowania temperatury odpowiednie do badań na poziomie architektury komputerów. Opisano metody ograniczania temperatury, które wymagają separacji obliczeń w czasie, czyli spowolnienia obliczeń, lub w przestrzeni, jako wykorzystanie wolnych zasobów w innej części procesora. Sprzętowe mechanizmy dynamicznego zarządzania temperaturą (DTM) charakteryzuje duża skuteczność w ograniczaniu temperatury i wysoka niezawodność, ale ich działanie wiąże się ze znaczącym spowolnieniem przetwarzania. Mechanizmy programowe, na podstawie informacji o przetwarzanych zadaniach, umożliwiają optymalizację wydajności. Postęp w dziedzinie wytwarzania układów scalonych powoduje wykładniczy wzrost liczby elementów logicznych w jednym układzie i wymaga efektywnego zarządzania zasobami procesora. Upowszechnienie procesorów wielordzeniowych, połączo-

ne ze zmniejszaniem wymiarów poszczególnych rdzeni wymaga modyfikacji mechanizmów służących ograniczaniu temperatury. Zapewnienie wysokiej wydajności całego systemu wymaga całościowej optymalizacji i prognozy temperatury.

W ramach pracy przeanalizowano możliwość wysokopoziomowego modelowania wydajności, zużycia energii i temperatury procesora wielordzeniowego na potrzeby mechanizmu zarządzania jego temperaturą. Opracowane zostały modele wiążące wydajność procesora ze zużywaną energią oraz opisujące zmienną aktywność procesora w czasie wykonywania poszczególnych programów. Porównanie wyników symulacji z pomiarami przeprowadzonymi na rzeczywistym komputerze potwierdziło ich poprawność. Powszechnie stosowanym mechanizmem służącym do ograniczania temperatury oraz zmniejszania zużycia energii jest dynamiczne skalowanie napięcia i częstotliwości (DVFS). Opracowano więc model wiążący wydajność i rozpraszaną moc z poziomami szybkości procesora co pozwala na dobranie maksymalnej szybkości pracy rdzeni bez przekraczania ustalonego limitu temperatury.

Aktywność procesora podczas przetwarzania programu może mieć charakter cykliczny. Wynika to z przekazywania sterowania w pętlach i wielokrotnego wykonywania tych samych fragmentów kodu. Na podstawie informacji o aktywności programu w przeszłości można przewidzieć moc rozpraszaną przez procesor w przyszłości oraz jego temperaturę. Prognoza temperatury procesora pozwala zapobiec niepożądanemu jej wzrostowi a jednocześnie zwiększyć szybkość przetwarzania jeśli prognozowany jest jej niski poziom. Z uwagi na szybkość zmian temperatury (zaobserwowano zmiany do 5°C w ciągu 10 ms) oraz znaczny narzut związany z wykorzystaniem niektórych mechanizmów dynamicznego zarządzania temperaturą potrzebna jest metoda która zapewnia długi okres prognozy. W niniejszej pracy przedstawiono metodę prognozy temperatury procesora na podstawie jego modelu termicznego oraz informacji o zachowaniu wykonywanych programów. Zaprezentowany algorytm prognozy pozwolił na zmniejszenie maksymalnego błędu oraz liczby błędnych prognoz w stosunku do prognozy na podstawie poprzedniego odczytu temperatury, przy niskiej złożoności obliczeniowej.

Wykorzystany w pracy model termiczny procesora jest modelem fizycznym co oznacza konieczność znajomość topografii procesora oraz oszacowanie właściwości fizycznych jego obudowy oraz radiatora. Przykładowo procesor wykorzystywany w eksperymentach składa się z dwóch modułów w jednej obudowie. Nieuwzględnienie wysokiej rezystancji termicznej pomiędzy nimi powodowało zawyżanie prognozy w jednej

części i jednoczesne zaniżanie w drugiej. Powodowało to błędy prognozy nawet o 6°C, jeśli tylko w jednym module były obciążone rdzenie. Znaczący wpływ na skuteczność prognozy temperatury ma dokładność wykorzystanego modelu termicznego. Błędne określenie jego parametrów prowadzi do ograniczenia użytecznego okresu prognozy. Wadą prezentowanego podejścia jest konieczność uprzedniego określenia właściwości termicznych procesora dla którego jest stosowana.

W pracy przeprowadzono eksperymentalną analizę skuteczności obniżania temperatury procesora przez mechanizmy skalowania napięcia i częstotliwości oraz migracji zadań. Eksperymenty przeprowadzono wykorzystując duży zestaw programów jedno i wielowątkowych oraz komputer z czterordzeniowym procesorem. Oba mechanizmy mają ograniczenia, wskutek czego w pewnych warunkach są albo nieskuteczne, albo powodują nieakceptowalnie duże spowolnienie wykonania programu. Przenoszenie zadania z gorącego na chłodny rdzeń pozwala wyrównać poziomy temperatury i obniżyć maksymalną temperaturę. Jest to jednak skuteczne jedynie przy znacznych różnicach temperatur i może powodować wyraźne spowolnienie przetwarzania w przypadku programów wielowątkowych. To ogranicza zastosowanie migracji wątków do przypadków kiedy tylko część rdzeni procesora jest zajętych.

Implementowane w nowoczesnych procesorach skalowanie napięcia i częstotliwości wiąże się z bardzo krótkim wstrzymaniem przetwarzania – rzędu 10 μs. Pozwala to na doraźne używanie tego mechanizmu i wprowadza minimalne opóźnienia. Niestety wpływ energii rozpraszanej przez sąsiednie rdzenie powoduje, że nawet obniżenie częstotliwości do minimalnego poziomu może nie wystarczyć do utrzymania temperatury poniżej wymaganego poziomu. Ponadto, w sytuacji gdy z powodu temperatury ograniczana jest częstotliwość taktowania jednego rdzenia podczas gdy pozostałe rdzenie są wolne, migracja byłaby bardziej efektywna. Wynika stąd, że maksymalizacja wydajności całego systemu komputerowego wymaga globalnej informacji o wszystkich rdzeniach i zadaniach w systemie. Przeszkodą może być rosnąca liczba rdzeni w przyszłych procesorach, co przełoży się na wzrost złożoności problemu.

Heurystyczny algorytm zarządzania pracą procesora przy ograniczeniu temperaturowym zaproponowany w pracy oparty jest na następujących założeniach. Aby zachować przydział zasobów zgodny z priorytetami zadań, należy w pierwszej kolejności redukować częstotliwość pracy rdzeni, na których uruchomione są zadania o niskim priorytecie. Strata wydajności związana z DVFS jest odwrotnie proporcjonalna do liczby chybień w pamięciach podręcznych i odwołań do pamięci głównej. Dlatego

należy maksymalizować całkowitą wydajność i w pierwszej kolejności obniżać częstotliwość pracy rdzeni, na których występuje relatywnie dużo chybień. Aby ograniczyć liczbę zbędnych migracji oraz liczbę okresów kiedy przekroczona jest temperatura maksymalna, należy wykorzystać prognozę temperatury procesora na podstawie znajomości zachowania programów i jego modelu termicznego. Maksymalizacja wydajności przetwarzania przy ograniczeniu temperaturowym może być zapewniona poprzez wyszukiwanie maksymalnej częstotliwości, która nie spowoduje przekroczenia $T_{\max}$.

Kontrolowanie temperatury procesora przy zapewnieniu odpowiedniego poziomu wydajności jest zagadnieniem bardzo złożonym. Trywialne przypadki, takie jak obciążenie procesora jednowątkowymi zadaniami o niskiej liczbie odwołań do pamięci, można relatywnie prosto zamodelować i przedstawić algorytm maksymalizujący wydajność przy zadanym ograniczeniu temperatury. W praktyce wydajność całego systemu zależy od bardzo wielu czynników i bez wiedzy o zachowaniu programów i organizacji procesora ograniczanie temperatury może powodować znaczącą utratę wydajności. Co więcej zachowanie termiczne procesora zależy od warunków w jakich on pracuje, a przede wszystkim od wydajności układu chłodzenia oraz temperatury otoczenia. Z tego powodu podejmowane decyzje dotyczące działania procesora powinny uwzględniać warunki otoczenia. Dodatkowo sterowanie pracą procesora, oprócz maksymalizacji wydajności i ograniczenia temperatury, powinno zapewniać minimalizację energii zużywanej przez cały system komputerowy. Aktywne chłodzenie procesora, oprócz kosztu samego systemu podnosi koszt jego użytkowania, zużywając znaczące ilości energii.

Wzrost znaczenia procesorów wielordzeniowych powoduje konieczność weryfikacji podejścia do zarządzania ich zasobami, również z powodu limitu ilości energii która może być rozproszona. Z powodu złożonego modelu programowego wielu aplikacji równoległych potrzebne jest określenie zachowanie termicznego nowych, równoległych architektur i programów. Wskazane jest przeprowadzenie eksperymentów oceniających zależność pomiędzy temperaturą a wydajnością w nowych procesorach ze współbieżną wielowątkowością (SMT) i ocena skuteczności kontrolowania temperatury za pomocą skalowania częstotliwości i odpowiedniego szeregowania zadań.

Mechanizmy oszczędzania energii są w powszechnym użyciu w systemach operacyjnych. Natomiast obsługa czujników temperatury ogranicza się zwykle do monitorowania i ostrzegania przed wysoką temperaturą. Biorąc pod uwagę wzajemny wpływ temperatury i wydajności przetwarzania, potrzebna jest do implementacja

DTM w systemie operacyjnym jako części mechanizmu szeregowania zadań. Pozwoli to wykorzystywać informację o temperaturze każdorazowo podczas wyboru kolejnego wątku do uruchomienia. Takie podejście będzie skutkowało minimalnymi stratami wydajności. Implementacja mechanizmu DTM w powszechnie używanym systemie operacyjnym zwiększyłaby również możliwość porównywania wyników przez środowisko zajmujące się tą tematyką. Jak pokazano w pracy, na podstawie informacji o przebiegu programu można prognozować temperaturę procesora w przyszłości co pozwala na zwiększenie jego wydajności przy ustalonym poziomie temperatury maksymalnej. Można przypuszczać, że implementacja metod dynamicznego zarządzania temperaturą w systemach operacyjnych pozwoli zwiększyć wydajność przetwarzania w wielu zastosowaniach.

Bibliografia

[1] H. F. Hamann, A. Weger, J. A. Lacey, Z. Hu, P. Bose, E. Cohen and J. Wakil, "Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements", *Solid-State Circuits, IEEE Journal of*, vol. 42, n. 1, pp. 56–65, jan. 2007.

[2] A.K. Coskun, J.L. Ayala, D. Atienza, T.S. Rosing and Y. Leblebici, "Dynamic thermal management in 3D multicore architectures", in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), 2009*, pp. 1410–1415, april 2009.

[3] Francisco Javier Mesa-Martinez, Ehsan K. Ardestani and Jose Renau, "Characterizing processor thermal behavior", in *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS '10, pp. 193–204, New York, NY, USA, 2010, ACM.

[4] Wei Huang, M. Allen-Ware, J.B. Carter, E. Cheng, K. Skadron and M.R. Stan, "Temperature-Aware Architecture: Lessons and Opportunities", *Micro, IEEE*, vol. 31, n. 3, pp. 82–86, may-june 2011.

[5] Joonho Kong, Sung Woo Chung and Kevin Skadron, "Recent Thermal Management Techniques for Microprocessors", *ACM Computing Survey*, 2011.

[6] N.H.E. Weste and D.F. Harris, *CMOS VLSI design: a circuits and systems perspective*, Addison-Wesley, 2005.

[7] Y.A. Cengel and Y. Feng, *Heat transfer: a practical approach*, McGraw-Hill New York, 2003.

[8] M. Pedram and S. Nazarian, "Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods", *Proceedings of the IEEE*, vol. 94, n. 8, pp. 1487–1501, aug. 2006.

[9] K. Mistry, C. Allen, C. Auth, B. Beattie, D. Bergstrom, M. Bost, M. Brazier, M. Buehler, A. Cappellani, R. Chau, C.-H. Choi, G. Ding, K. Fischer, T. Ghani, R. Grover, W. Han, D. Hanken, M. Hattendorf, J. He, J. Hicks, R. Huessner, D. Ingerly, P. Jain, R. James, L. Jong, S. Joshi, C. Kenyon, K. Kuhn, K. Lee, H. Liu, J. Maiz, B. Mclntyre, P. Moon, J. Neirynck, S. Pae, C. Parker, D. Parsons, C. Prasad, L. Pipes, M. Prince, P. Ranade, T. Reynolds, J. Sandford, L. Shifren, J. Sebastian, J. Seiple, D. Simon, S. Sivakumar, P. Smith, C. Thomas, T. Troeger, P. Vandervoorn, S. Williams and K. Zawadzki, "A 45nm Logic Technology with High-k+Metal Gate Transistors, Strained Silicon, 9 Cu Interconnect Layers, 193nm Dry Patterning, and 100Pb-free Packaging", in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pp. 247–250, dec. 2007.

[10] "Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors", 2008.

[11] L. Gwennap, "Sandy Bridge spans generations", *Microprocessor Report*, vol. 9, n. 27, pp. 10–01, 2010.

[12] JEDEC Publication JEP122E, "Failure Mechanisms and Models for Semiconductor Devices", 2009.

[13] J. Srinivasan, S.V. Adve, P. Bose and J.A. Rivers, "The case for lifetime reliability-aware microprocessors", in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pp. 276–287, june 2004.

[14] Tajana Simunic Rosing, Kresimir Mihic and Giovanni De Micheli, "Power and reliability management of SoCs", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, n. 4, pp. 391–403, 2007.

[15] Zhijian Lu, Wei Huang, Mircea R. Stan, Kevin Skadron and John Lach, "Interconnect lifetime prediction for reliability-aware systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, n. 2, pp. 159–172, 2007.

[16] E. Karl, D. Blaauw, D. Sylvester and T. Mudge, "Multi-mechanism reliability modeling and management in dynamic systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, n. 4, pp. 476–487, 2008.

[17] J. Srinivasan, S.V. Adve, P. Bose and J.A. Rivers, "Exploiting structural duplication for lifetime reliability enhancement", *ACM SIGARCH Computer Architecture News*, vol. 33, n. 2, pp. 520–531, 2005.

[18] A.K. Coskun, T.S. Rosing, Y. Leblebici and G. De Micheli, "A simulation methodology for reliability analysis in multi-core SoCs", in *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pp. 95–99. ACM, 2006.

[19] A.K. Coskun, R. Strong, D.M. Tullsen and T.S. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors", in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pp. 169–180. ACM New York, NY, USA, 2009.

[20] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando and M.R. Stan, "Differentiating the roles of IR measurement and simulation for power and temperature-aware design", in *Proceedings of the 2009 International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 1–10, 2009.

[21] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations", *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 83–94, 2000.

[22] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill and David A. Wood, "The gem5 simulator", *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, August 2011.

[23] T. Austin, E. Larson and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling", *COMPUTER*, pp. 59–67, 2002.

[24] S. Gunther, F. Binns, D. M. Carmean and J. C. Hall, "Managing the Impact of Increasing Microprocessor Power Consumption", *Intel Technology Journal*, vol. 5, n. 1, 2001.

[25] S. Thoziyoor, N. Muralimanohar, J. Ahn and N.P. Jouppi, "CACTI 5.1", Technical report, HPL-2008-20, HP Labs, 2008.

[26] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen and N.P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures", in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480. ACM, 2009.

[27] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron and M.R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 14, n. 5, pp. 501–513, 2006.

[28] W. Huang, K. Sankaranarayanan, R.J. Ribando, M.R. Stan and K. Skadron, "An Improved Block-Based Thermal Model in HotSpot 4.0 with Granularity Considerations", *In Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking (WDDD), in conjunction with ISCA-34*, 2007.

[29] P. Michaud and Y. Sazeides, "ATMI: analytical model of temperature in microprocessors", in *Third Annual Workshop on Modeling, Benchmarking and Simulation*, 2007.

[30] M. Monchiero, R. Canal and A. Gonzalez, "Power/Performance/Thermal Design-Space Exploration for Multicore Architectures", *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, n. 5, pp. 666–681, 2008.

[31] P. Chaparro, J. Gonzalez, G. Magklis, C. Qiong, A. Gonzalez and B. Upc, "Understanding the thermal implications of multi-core architectures", *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, n. 8, pp. 1055–1065, 2007.

[32] Pu Liu, Hang Li, Lingling Jin, Wei Wu, Sheldon X.-D. Tan and Jun Yang, "Fast Thermal Simulation for Runtime Temperature Tracking and Management", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, n. 12, pp. 2882–2893, dec. 2006.

[33] T.J. Eguia, S.X.-D. Tan, Ruijing Shen, Duo Li, E.H. Pacheco, M. Tirumala and Lingli Wang, "General Parameterized Thermal Modeling for High-Performance Microprocessor Design", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, n. 2, pp. 211–224, feb. 2012.

[34] Y. Han, I. Koren and CM Krishna, "Temptor: A lightweight runtime temperature monitoring tool using performance counters", in *In Proceedings of the Third Workshop on Temperature Aware Computer Systems (TACS'06)*, June 2006.

[35] Dongsheng Ma and R. Bondade, "Enabling Power-Efficient DVFS Operations on Silicon", *Circuits and Systems Magazine, IEEE*, vol. 10, n. 1, pp. 14–30, quarter 2010.

[36] P. Dadvar and K. Skadron, "Potential thermal security risks", in *Semiconductor Thermal Measurement and Management Symposium, 2005 IEEE Twenty First Annual IEEE*, pp. 229–234, march 2005.

[37] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors", in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pp. 171 –182, 2001.

[38] S. Ghosh, Jung-Hwan Choi, P. Ndai and K. Roy, "O2C: occasional two-cycle operations for dynamic thermal management in high performance in-order microprocessors", in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, pp. 189–192, aug. 2008.

[39] K. Skadron, T. Abdelzaher and M.R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management", in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, page 17. IEEE Computer Society Washington, DC, USA, 2002.

[40] Kevin Skadron, "Hybrid Architectural Dynamic Thermal Management", in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), 2004*, page 10010, Washington, DC, USA, 2004, IEEE Computer Society.

[41] A. Kumar, L. Shang, L.S. Peh and N.K. Jha, "HybDTM: a coordinated hardware-software approach for dynamic thermal management", in *Proceedings of the 43rd annual Conference on Design Automation*, pp. 548–553. ACM New York, NY, USA, 2006.

[42] J. Donald and M. Martonosi, "Leveraging simultaneous multithreading for adaptive thermal control", in *Proc. of the Second Workshop on Temperature-Aware Computer Systems*. Citeseer, 2005.

[43] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration", *International Symposium on Computer Architecture: Proceedings of the 33rd annual International Symposium on Computer Architecture*, vol. 17, n. 21, pp. 78–88, 2006.

[44] M. Kadin, S. Reda and A. Uht, "Central vs. distributed dynamic thermal management for multi-core processors: which one is better?", in *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pp. 137–140. ACM, 2009.

[45] R. Mukherjee and S.O. Memik, "Physical aware frequency selection for dynamic thermal management in multi-core systems", in *Proceedings of the 2006 IEEE/ACM International conference on Computer-aided design*, pp. 547–552. ACM Press New York, NY, USA, 2006.

[46] M. Kadin and S. Reda, "Frequency and voltage planning for multi-core processors under thermal constraints", in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pp. 463–470, oct. 2008.

[47] Michael Kadin and Sherief Reda, "Frequency planning for multi-core processors under thermal constraints", in *ISLPED '08: Proceeding of the thirteenth International Symposium on Low Power Electronics and Design*, pp. 213–216, New York, NY, USA, 2008, ACM.

[48] Mohamed Gomaa, Michael D. Powell and T. N. Vijaykumar, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system", in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS-XI, pp. 260–270, New York, NY, USA, 2004, ACM.

[49] A. Kumar, Li Shang, Li-Shiuan Peh and N.K. Jha, "System-Level Dynamic Thermal Management for High-Performance Microprocessors", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, n. 1, pp. 96–108, jan. 2008.

[50] H. Jung, P. Rong and M. Pedram, "Stochastic modeling of a thermally-managed multi-core system", in *Proceedings of the 45th annual conference on Design automation*, pp. 728–733. ACM New York, NY, USA, 2008.

[51] Hwisung Jung and M. Pedram, "A stochastic local hot spot alerting technique", in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pp. 468–473, march 2008.

[52] Donghwa Shin, Jihun Kim, Naehyuck Chang, Jinhang Choi, Sung Woo Chung and Eui-Young Chung, "Energy-optimal dynamic thermal management for green computing", in *Computer-Aided Design – Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pp. 652–657, nov. 2009.

[53] O. Khan and S. Kundu, "Hardware/software co-design architecture for thermal management of chip multiprocessors", in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), 2009*, pp. 952–957, april 2009.

[54] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides and T. Constantinou, "A study of thread migration in temperature-constrained multicores", *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, n. 2, 2007.

[55] R. Rao, S. Vrudhula and K. Berezowski, "Analytical results for design space exploration of multi-core processors employing thread migration", in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, pp. 229–232, aug. 2008.

[56] V. Hanumaiah, R. Rao, S. Vrudhula and K.S. Chatha, "Throughput Optimal Task Allocation under Thermal Constraints for Multi-core Processors", in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp. 776–781, july 2009.

[57] Jun Yang, Xiuyi Zhou, M. Chrobak, Youtao Zhang and Lingling Jin, "Dynamic Thermal Management through Task Scheduling", in *Performance Analysis of Systems and Software, 2008. ISPASS 2008. IEEE International Symposium on*, pp. 191–201, april 2008.

[58] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger and Pradip Bose, "Thermal-aware task scheduling at the system software

level", in *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pp. 213–218, New York, NY, USA, 2007, ACM.

[59] L. Xia, Y. Zhu, J. Yang, J. Ye and Z. Gu, "Implementing a Thermal-Aware Scheduler in Linux Kernel on a Multi-Core Processor", *The Computer Journal*, vol. 53, n. 7, pp. 895–903, 2010.

[60] F. Zanini, D. Atienza, L. Benini and G. De Micheli, "Multicore thermal management with model predictive control", in *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pp. 711–714. IEEE, 2009.

[61] A. Bartolini, M. Cacciari, A. Tilli and L. Benini, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores", in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–6, march 2011.

[62] Peter Bailis, Vijay Janapa Reddi, Sanjay Gandhi, David Brooks and Margo Seltzer, "Dimetrodon: Processor-level preventive thermal management via idle cycle injection", in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pp. 89–94, june 2011.

[63] I. Yeo, C.C. Liu and E.J. Kim, "Predictive dynamic thermal management for multicore systems", in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pp. 734–739, 2008.

[64] A.K. Coskun, T.S. Rosing and K.C. Gross, "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, n. 10, pp. 1503–1516, oct. 2009.

[65] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, G. De Micheli and E. LSI, "Temperature control of high-performance multi-core platforms using convex optimization", pp. 110–115, 2008.

[66] J.S. Lee, K. Skadron and S.W. Chung, "Predictive Temperature-Aware DVFS", *IEEE Transactions on Computers*, pp. 127–133, 2010.

[67] R.Z. Ayoub and T.S. Rosing, "Predict and act: dynamic thermal management for multi-core processors", in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 99–104. ACM, 2009.

[68] Vinay Hanumaiah, Sarma Vrudhula and Karam S. Chatha, "Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control", in *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 310–313, New York, NY, USA, 2009, ACM.

[69] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors", in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 537–542. IEEE Press Piscataway, NJ, USA, 2008.

[70] Xiuyi Zhou, Jun Yang, Marek Chrobak and Youtao Zhang, "Performance-aware thermal management via task scheduling", *ACM Transactions on Architecture and Code Optimization*, vol. 7, n. 1, pp. 1–31, 2010.

[71] Yefu Wang, Kai Ma and Xiaorui Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation", in *Proceedings of the 36th annual International Symposium on Computer architecture*, ISCA '09, pp. 314–324, New York, NY, USA, 2009, ACM.

[72] O. Khan, "A Hardware/Software Co-Design Architecture for Thermal, Power, and Reliability Management in Chip Multiprocessors", *Open Access Dissertations*, page 169, 2010.

[73] Jayanth Srinivasan and Sarita V. Adve, "Predictive dynamic thermal management for multimedia applications", in *Proceedings of the 17th annual International Conference on Supercomputing*, ICS '03, pp. 109–120, New York, NY, USA, 2003, ACM.

[74] Yang Ge and Qinru Qiu, "Dynamic thermal management for multimedia applications using machine learning", in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pp. 95–100, june 2011.

[75] H. Jung and M. Pedram, "Stochastic Dynamic Thermal Management: A Markovian Decision-based Approach", in *ICCD 2006. International Conference on Computer Design, 2006.*, pp. 452–457, 2006.

[76] ITRS, "International Technology Roadmap for Semiconductors – Executive Summary", Technical report, ITRS, 2011.

[77] S. Borkar, "Thousand Core Chips: A Technology Perspective", in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pp. 746–749, june 2007.

[78] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom et al., "A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS", in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 108–109. IEEE, 2010.

[79] S. Sharifi, A.K. Coskun and T.S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs", in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 873–878, jan. 2010.

[80] H.P. Hofstee, "Power Efficient Processor Architecture and The Cell Processor", in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pp. 258–262. IEEE Computer Society Washington, DC, USA, 2005.

[81] Xin Li, Yuchun Ma and Xianlong Hong, "A novel thermal optimization flow using incremental floorplanning for 3D ICs", in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pp. 347–352, jan. 2009.

[82] Marek Chrobak, Christoph Dürr, Mathilde Hurand and Julien Robert, "Algorithms for Temperature-Aware Task Scheduling in Microprocessor Systems", in *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pp. 120–130, Berlin, Heidelberg, 2008, Springer-Verlag.

[83] W. Huang, M.R. Stan, K. Sankaranarayanan, R.J. Ribando and K. Skadron, "Many-Core Design from a Thermal Perspective: Extended Analysis and Results", Technical report, Technical Report CS-2008-05, University of Virginia, Computer Science Department, April 2008.

[84] K. Sankaranarayanan, W. Huang, M.R. Stan, H. Haj-Hariri, R.J. Ribando and K. Skadron, "Granularity of microprocessor thermal management: a technical report", Technical report, CS-2009-03, University of Virginia Department of Computer Science, 2009.

[85] A.K. Coskun, T.S. Rosing and K.C. Gross, "Proactive temperature management in MPSoCs", in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, pp. 165–170, aug. 2008.

[86] R. Cochran and S. Reda, "Consistent runtime thermal prediction and control through workload phase detection", in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 62–67, june 2010.

[87] R. Rao and S. Vrudhula, "Fast and accurate prediction of the steady-state throughput of multicore processors under thermal constraints", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, n. 10, pp. 1559–1572, 2009.

[88] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando and M.R. Stan, "Exploring the thermal impact on manycore processor performance", in *Semiconductor Thermal Measurement and Management Symposium, 2010. SEMI-THERM 2010. 26th Annual IEEE*, pp. 191–197. IEEE, 2010.

[89] M. Jahre, M. Grannaes and L. Natvig, "A Quantitative Study of Memory System Interference in Chip Multiprocessor Architectures", in *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on*, pp. 622–629, june 2009.

[90] A.R. Alameldeen and D.A. Wood, "IPC Considered Harmful for Multiprocessor Workloads", *Micro, IEEE*, vol. 26, n. 4, pp. 8–17, july-aug. 2006.

[91] K. Sankaranarayanan, S. Velusamy, M. Stan and K. Skadron, "A case for thermal-aware floorplanning at the microarchitectural level", *Journal of Instruction-Level Parallelism*, vol. 7„ pp. 1–16, 2005.

[92] S. Reda, "Thermal and Power Characterization of Real Computing Devices", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, n. 2, pp. 76–87, june 2011.

[93] Zhenyu Qi, B.H. Meyer, Wei Huang, R.J. Ribando, K. Skadron and M.R. Stan, "Temperature-to-power mapping", in *Computer Design (ICCD), 2010 IEEE International Conference on*, pp. 384–389, oct. 2010.

[94] Hyung Beom Jang, Jinhang Choi, Ikroh Yoon, Sung-Soo Lim, Seungwon Shin, Naehyuck Chang and Sung Woo Chung, "Exploiting application-dependent ambient temperature for accurate architectural simulation", in *Computer Design (ICCD), 2010 IEEE International Conference on*, pp. 502–508, oct. 2010.

[95] Jeonghee Shin, J.A. Darringer, Guojie Luo, A.J. Weger and C.L. Johnson, "Early chip planning cockpit", in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–4, march 2011.

[96] K. Skadron, M.R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan and D. Tarjan, "Temperature-aware microarchitecture", *ACM SIGARCH Computer Architecture News*, vol. 31, n. 2, pp. 2–13, 2003.

[97] R. Ayoub, K. Indukuri and T.S. Rosing, "Temperature Aware Dynamic Workload Scheduling in Multisocket CPU Servers", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, n. 9, pp. 1359–1372, sept. 2011.

[98] Intel, "Intel® Core^{TM}2 Extreme Processor QX9000$^{\Delta}$ Series, Intel Core^{TM}2 Quad Processor Q9000$^{\Delta}$, Q9000S$^{\Delta}$, Q8000$^{\Delta}$, and Q8000S$^{\Delta}$ Series", august 2009.

[99] Kyeong-Jae Lee, K. Skadron and W. Huang, "Analytical model for sensor placement on microprocessors", in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD 2005*, pp. 24–27, oct. 2005.

[100] Jieyi Long, Seda Ogrenci Memik, Gokhan Memik and Rajarshi Mukherjee, "Thermal monitoring mechanisms for chip multiprocessors", *ACM Transactions on Architecture and Code Optimization*, vol. 5, pp. 9:1–9:33, September 2008.

[101] K.J. Lee and K. Skadron, "Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors", in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 232–232, 2005.

[102] S. Sharifi and T.S. Rosing, "Accurate Direct and Indirect On-Chip Temperature Sensing for Efficient Dynamic Thermal Management", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, n. 10, pp. 1586–1599, oct. 2010.

[103] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A & 3B): System Programming Guide", may 2011.

[104] M. Berktold and T. Tian, "CPU Monitoring With DTS/PECI", 2009, Intel whitepaper.

[105] T. Mudge, "Power: A First-Class Architectural Design Constraint", *COMPUTER*, pp. 52–58, 2001.

[106] Weiping Liao, Lei He and K.M. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, n. 7, pp. 1042–1053, july 2005.

[107] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam and Doug Burger, "Dark silicon and the end of multicore scaling", *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 365–376, June 2011.

[108] W. Bircher, J. Law, M. Valluri and L.K. John, "Effective use of performance monitoring counters for run-time prediction of power", *University of Texas at Austin Technical Report TR-041104-01*, 2004.

[109] John L. Hennessy and David A. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann, 2007.

[110] F.J. Mesa-Martinez, M. Brown, J. Nayfach-Battilana and J. Renau, "Measuring performance, power, and temperature from real processors", in *Proceedings of the 2007 workshop on Experimental computer science*. ACM New York, NY, USA, 2007.

[111] "SPEC CPU Benchmark Suites", 2010, http://www.spec.org/cpu/index.html.

[112] C. Bienia, S. Kumar, J.P. Singh and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications", in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 72–81. ACM, 2008.

[113] Wei Huang, K. Sankaranarayanan, K. Skadron, R.J. Ribando and M.R. Stan, "Accurate, Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model", *Computers, IEEE Transactions on*, vol. 57, n. 9, pp. 1277–1288, sept. 2008.

[114] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically characterizing large scale program behavior", *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 45–57, 2002.

[115] M.J. Hind, V.T. Rajan and P.F. Sweeney, "Phase shift detection: A problem classification", *IBM Research Report RC*, vol. 22887, 2003.

[116] AS Dhodapkar and JE Smith, "Comparing program phase detection techniques", in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp. 217–227, 2003.

[117] A.S. Dhodapkar and J.E. Smith, "Managing multi-configuration hardware via dynamic working set analysis", *ACM SIGARCH Computer Architecture News*, vol. 30, n. 2, pp. 233–244, 2002.

[118] R. Joseph, M. Martonosi and Z. Hu, "Spectral analysis for characterizing program power and performance", in *2004 IEEE ISPASS International Symposium on Performance Analysis of Systems and Software*, pp. 151–160, 2004.

[119] B. Wojciechowski, K.S. Berezowski, P. Patronik and J. Biernat, "Fast and accurate thermal simulation and modelling of workloads of many-core processors", in *Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on*, pp. 1–6, sept. 2011.

[120] Hiroshi Sasaki, Yoshimichi Ikeda, Masaaki Kondo and Hiroshi Nakamura, "An intra-task DVFS technique based on statistical analysis of hardware events",

in *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, pp. 123–130, New York, NY, USA, 2007, ACM.

[121] J. Clabes, J. Friedrich, M. Sweet, J. Dilullo, S. Chu, D. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd et al., "Design and implementation of the POWER5 microprocessor", in *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pp. 56–57, 2004.

[122] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang and R. Kumar, "An Integrated Quad-Core Opteron Processor", in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pp. 102–103, feb. 2007.

[123] K. Skadron, MR Stan, W. Huang, S. Velusamy, K. Sankaranarayanan and D. Tarjan, "Temperature-aware microarchitecture: Extended discussion and results", *University of Virginia, Dept. of Computer Science, Technicial Report CS-2003*, 2003.

[124] Yufu Zhang and Ankur Srivastava, "Accurate temperature estimation using noisy thermal sensors", in *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pp. 472–477, New York, NY, USA, 2009, ACM.

[125] Wei Wu, Lingling Jin, Jun Yang, Pu Liu and Sheldon X.-D. Tan, "Efficient power modeling and software thermal sensing for runtime temperature monitoring", *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, pp. 25–29, May 2007.

[126] R. Love, *Linux kernel development*, Addison-Wesley Professional, 2009.

[127] Panos J. Antsaklis and Anthony N. Michel, *A Linear Systems Primer*, Birkhaüser Basel, 1. Edition, 2007.

[128] Canturk Isci, Gilberto Contreras and Margaret Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management", in *39th Annual IEEE/ACM International Symposium onMicroarchitecture, 2006, MICRO-39.*, pp. 359–370, dec. 2006.

[129] T. Sherwood, S. Sair and B. Calder, "Phase tracking and prediction", in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp. 336–347, june 2003.

[130] C. Isci, A. Buyuktosunoglu, C.Y. Cher, P. Bose and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget", in *International Symposium on Microarchitecture: Proceedings of the 39 th Annual IEEE/ACM International Symposium on Microarchitecture*, volume 9, pp. 347–358, 2006.

[131] O. Khan and S. Kundu, "A framework for predictive dynamic temperature management of microprocessor systems", in *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pp. 258–263, nov. 2008.

[132] H. Hanson, S.W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson and J. Rubio, "Thermal response to DVFS: analysis with an Intel Pentium M", in *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pp. 219–224, aug. 2007.

[133] Youngwoo Ahn, Inchoon Yeo and R. Bettati, "Efficient calibration of thermal models based on application behavior", in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pp. 41–46, oct. 2009.

[134] D. Koufaty and D.T. Marr, "Hyperthreading technology in the netburst microarchitecture", *Micro, IEEE*, vol. 23, n. 2, pp. 56–65, 2003.

[135] P. Kongetira, K. Aingaran and K. Olukotun, "Niagara: A 32-way multithreaded SPARC processor", *IEEE Micro-Institute of Electrical and Electronics Engineers*, vol. 25, n. 2, pp. 21–29, 2005.

[136] T. Fischer, S. Arekapudi, E. Busta, C. Dietz, M. Golden, S. Hilker, A. Horiuchi, K.A. Hurd, D. Johnson, H. McIntyre, S. Naffziger, J. Vinh, J. White and K. Wilcox, "Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU", in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pp. 78–80, feb. 2011.

[137] D. Citron, J. Hennessy, D. Patterson and G.S. Sohi, "The use and abuse of SPEC: An ISCA panel", 2003.