Analiza i modelowanie wydajności obliczeń

Lab 6. Wielowątkowość i odwzorowanie wątków na elementy sprzętowe

Cel: Przetestowanie możliwości przypisywania wątków do mikroprocesorów i rdzeni (oraz wątków sprzętowych) oraz sprawdzenie wpływu przypisania na wydajność

Wstęp

1. W ramach laboratorium badany będzie wpływ przypisania wątków na wydajność wykonania dla dwóch programów dostępnych w paczkach na stronie przedmiotu:
    1. *latency_throughput_multithreaded_vector_flops* (będący specjalną wielowątkową wersją programu *latency_throughput_vector_flops* z laboratorium nr 3) – jako przykład wykonania wielowątkowego, dla którego czas wykonania i wydajność są w pełni determinowane przez możliwości przetwarzania potoków zmiennoprzecinkowych w rdzeniach
    2. *random_pointer_chasing_mth* (będący wielowątkowym wariantem kodu z laboratorium 5) – jako przykład obliczeń wielowątkowych, dla których wydajność i czas wykonania są determinowane przez szybkość pobierania danych z pamięci i jednocześnie takich, że organizacja obliczeń utrudnia (lub wręcz uniemożliwia) wykorzystanie możliwości ukrywania opóźnienia w dostępie do pamięci oferowanych przez sprzęt (z rozmaitych powodów analizowanych w laboratorium 5)
2. Zadania realizowane dla poszczególnych programów, będą polegały na obserwacji wydajności i czasu wykonania obliczeń wielowątkowych, z jawnym przypisaniem wątków do procesorów logicznych (a w konsekwencji także rdzeni i mikroprocesorów). Wykorzystane w tym celu będą:
    - polecenie *numactl* z opcjami umożliwiającymi przydzielenie wątków konkretnym procesorom logicznym
    - ***możliwości kompilatorów (konkretnie kompilatora icc) sterowania przypisaniem wątków do procesorów logicznych za pomocą specjalnych zmiennych środowiskowych***
3. Eksperymenty obliczeniowe z wykorzystaniem wielowątkowości przeprowadzane mogą być dla zmiennych liczb wątków. Przy obliczeniach w trakcie zajęć na serwerze Honorata mogą wystąpić zaburzenia pomiarów wydajności dla większej liczby wątków, spowodowane pracą wielu osób na pojedynczej maszynie. Dla lepszej jakości wyników (i w konsekwencji poprawności wyciąganych z nich wniosków) zadania obowiązkowe ograniczone są do małej liczby wątków (Honorata – 1,2,4), a realizacja eksperymentów dla większej liczby wątków pozostawiona jest do przeprowadzenia indywidualnego w czasie mniejszego obciążenia maszyny. Jako efekt samych zajęć uznawane będzie wypełnienie odpowiednich tabelek definiowanych w kolejnych punktach dla wybranej liczby wątków (Honorata, co najmniej – 1,2,4)
4. Przy przypisywaniu wątków (programowych) do procesorów logicznych (wątków sprzętowych) stosowanych będzie kilka wzorców i odpowiadających im opcji polecenia *numactl* (w przykładach poniżej założona jest architektura serwera Honorata). Polecenie *numactl* pozwala wskazać procesory logiczne, na których wykonywane będą obliczenia (liczba wskazanych procesorów logicznych jest niezależna od liczby wątków w programie (można nawet przypisać wszystkie wątki jednemu procesorowi logicznemu) – wskazuje tylko możliwości przypisania, choć istotna jest także kolejność wskazanych procesorów; szczegóły *man numactl*). Wzorcami tymi mogą być:
    1. Wzorzec 1 – obsadzanie kolejnych procesorów logicznych, np.:
        - *numactl -C 0,24,1,25,2,26,3,27,...,12,36,13,37,14,38,15,39,...*
    2. Wzorzec 2 – obsadzanie kolejnych rdzeni, a dopiero po wykorzystaniu wszystkich rdzeni wykorzystanie dodatkowych procesorów logicznych przypisanych do tego samego rdzenia

- *numactl -C 0,1,2,3,...,12,13,14,15,...,24,25,26,27,...,36,37,38,39,...*

3. Wzorzec 3 – obsadzanie kolejnych gniazd (*socket*), po wyczerpaniu wszystkich gniazd powrót do kolejnego procesora logicznego pierwszego gniazda, itd.
   - *numactl -C 0,12,24,36,1,13,25,37,2,14,26,38,3,15,27,39,...*
4. Wzorzec 4 – obsadzanie kolejnych gniazd, po wyczerpaniu wszystkich gniazd powrót do kolejnego rdzenia fizycznego pierwszego gniazda, itd.
   - *numactl -C 0,12,1,13,2,14,3,15,...,24,36,25,37,26,38,27,39,...*

Kolejne kroki realizowane w ramach laboratorium:

2. **Zadanie 1. Uzyskanie danych o architekturze komputera/serwera i ich analiza.**
   1. Uruchom w terminalu polecenie wypisania zawartości pliku */proc/cpuinfo* . Przeanalizuj wydruk pojawiający się na ekranie – w szczególności numerację poszczególnych rdzeni fizycznych i procesorów logicznych (w przypadku istnienia jakiejś formy sprzętowej wielowątkowości - SMT, *simultaneous multithreading, hyperthreading* itp. - numeracje te są różne). Dla każdego z widzianych przez system operacyjny logicznych procesorów (oznaczających wątki sprzętowe, ID wyświetlany jest w linii oznaczanej *processor),* linia *physical id* oznacza numer gniazda (mikroprocesora), a linia *core id* zawiera numer rdzenia fizycznego (inne interesujące z punktu widzenia wydajności dane zapisywane w */proc/cpuinfo* to np. częstotliwość pracy w MHz, rozmiar pamięci podręcznej L3, liczba fizycznych rdzeni (w pojedynczym mikroprocesorze (gnieździe) *cpu cores*), liczba procesorów logicznych (także dla jednego gniazda, *siblings),* itp.). Wypisz w tabeli o formacie podanym poniżej odczytane wartości (dla kilkunastu wybranych procesorów logicznych (wątków sprzętowych, *hardware threads, HWThread*)

log_proc ID	0	1	2	3	4	5	...												
core ID																			
CPU ID																			

Numeracja logicznych procesorów jest istotna z punktu widzenia wydajności, gdyż posługuje się nią system operacyjny przy przypisywaniu kolejnych wątków systemowych do kolejnych procesorów logicznych (utożsamianych często z wątkami sprzętowymi). W przypadku mikroprocesorów z SMT, zazwyczaj najpierw numerowane są procesory logiczne (wątki sprzętowe) po jednym na fizyczny rdzeń, a dopiero potem wątki sprzętowe korzystające z SMT.

   2. Uruchom w terminalu polecenie *numactl -H* (*numactl* nie zawsze znajduje się w standardowych dystrybucjach Linuksa, powinno jednak być łatwe znalezienie odpowiedniego pakietu dla konkretnej wersji systemu). Zgodnie z dokumentacją (*man numactl*) *numactl* korzysta z informacji w */proc/cpuinfo* . Polecenie uruchomione z opcją *-H* wypisuje na ekranie nie tylko numery kolejnych procesorów logicznych, ale także przyporządkowanie fizycznych elementów pamięci DRAM procesorom logicznym (a w efekcie także wątkom pracującym na procesorach logicznych). W przypadku komputerów wieloprocesorowych (pojedyncze gniazdo mikroprocesora określane jako *node*), *node distances* odnosi się do faktu stosowania w maszynie pamięci o niejednorodnym dostępie (NUMA), gdzie układy DRAM są przypisywane konkretnym gniazdom (za pomocą kanałów pamięci), a dostęp do własnych układów jest szybszy niż dostęp do układów przypisanych innym gniazdom (dostęp odbywa się przez pośredniczące kanały połączeń miedzy gniazdami). Różnice w czasie dostępu do konkretnych pul pamięci znajdują odbicie w wartościach *node distances*.
   3. Bardziej szczegółowe informacje o przypisaniu pamięci wątkom można uzyskać korzystając z pakietu *likwid.* Uruchomienie polecenia *likwid-topology* pozwala na wyświetlenie przypisania procesorom logicznym (i wykonywanych na nich wątkom)

modułów pamięci podręcznej – konkretnie *likwid-topology* wypisuje grupy wątków sprzętowych (*HWThread*) korzystających wspólnie z jednego modułu pamięci podręcznej odpowiedniego poziomu.

4. Ostatnim z nowych wykorzystywanych w laboratorium narzędzi jest program *htop*. Jest to wersja polecenia *top* wzbogacona o wypisywanie na ekranie obciążenia konkretnych procesorów logicznych (graficzna reprezentacja procentu zajętości czasu CPU)

3. Zadanie 2. Badanie wpływu przypisania wątków do rdzeni dla programu, którego czas wykonania zależny jest od czasu przetwarzania operacji zmiennoprzecinkowych przez potoki rdzeni procesorów

1. Wykorzystany będzie program *latency_throughput_multithreaded_vector_flops* z laboratorium nr 3 (obliczenia można przeprowadzać w katalogu lab 3, ewentualnie skopiować paczkę *latency_throughput_flops.tgz* do katalogu lab 6 i tam rozpakować
    1. kod jest wielowątkową wersją programu *latency_throughput_vector_flops* , w której obliczenia na wyrazach tablic przeprowadzają odrębne wątki (wątki tworzone są za pomocą interfejsu *pthreads* i wszystkie wykonują tę samą funkcję zawierającą obliczenia)
    2. liczbą wątków można sterować za pomocą parametru NO_TH definiowanego na początku pliku (wymaga to rekompilacji kodu po każdej zmianie)
        1. definicję w pliku można także zakomentować i przekazywać argument podczas kompilacji (np. *icc -c -O3 -march=core-avx2 -DNO_TH=2 ....*)
    3. należy wybrać jako kompilator i linker *icc*, nie zapominając o włączeniu w *Makefile* opcji użycia rozkazów wektorowych *-march=core-avx2*
    4. przed badaniem wersji wielowątkowej należy sprawdzić czy program w wersji jednowątkowej (NO_TH równe 1) uzyskuje co najmniej 90% wydajności teoretycznej pojedynczego rdzenia (dla Honoraty ok. 44 Gflop/s, przy częstotliwości pracy ok. 2.8 GHz - do sprawdzenia np. za pomocą *perf stat*), m.in. rozmiar tablic w programie powinien być optymalny (wskazówka dla Honoraty: #define LOCAL_SIZE 16)
    5. czas i wydajność dla obliczeń jednowątkowych należy zapisać w celu porównania z parametrami wykonania wielowątkowego
    6. jako czas wykonania i wydajność należy zawsze brać wyniki z bloku zatytułowanego:

       Main program - calculations based on local measurements (no system overhead)

       wyniki te uzyskane są poprzez zmierzenie czasu wykonywania obliczeń z pominięciem czasu tworzenia wątków i innych narzutów (np. zastosowanej rozgrzewki)
        1. pominięcie narzutów ma na celu lepsze uchwycenie wpływu przypisania wątków do rdzeni na wydajność
        2. reprezentatywność czasu obliczeń zapewnia wykorzystanie bariery bezpośrednio przed obliczeniami i po obliczeniach, dla których mierzony jest czas

[tylko wersja w pełni wykorzystująca możliwości systemu, pozwala łatwo zaobserwować różnice wynikające z różnego przypisania wątków do rdzeni]

[dla serwera Honorata pełne możliwości to 16 flop/takt, co daje różne wartości w Gflop/s zależnie od częstotliwości pracy rdzenia]

2. Skompiluj wersję czterowątkową programu *latency_throughput_multithreaded_vector_flops*
3. Uruchom otrzymany program dla różnych sposobów przypisania wątków do rdzeni i zanotuj uzyskaną wydajność w tabeli
    1. każdorazowo należy zastosować strategię unikania wykonywania obliczeń na tych samych rdzeniach przez różne osoby (szczegóły na zajęciach)
    2. program *latency_throughput_multithreaded_vector_flops* wypisuje czas obliczeń i wydajność w Gflop/s

       (dodatkowo można uruchamiać obliczenia w ramach narzędzia *perf stat* i wykorzystując podaną częstotliwość pracy rdzenia obliczać wydajność w flop/takt – jest to dokładniejsze, ale także przybliżone, ze względu na

uwzględnianie przez *perf stat* całego programu; w pełni precyzyjne obliczenia wydajności w flop/takt umożliwia zastosowanie interfejsu PAPI)

3. jak zwykle należy obliczenia uruchomić kilka razy i wybrać przypadek z najkrótszym czasem obliczeń i najwyższą wydajnością
4. należy przetestować następujące warianty przypisania (każdorazowo poprzez zastosowanie odpowiedniej wersji polecenia *numactl* – użyty wariant należy odnotować w sprawozdaniu):
    1. wszystkie wątki do jednego procesora logicznego
    2. wszystkie wątki do jednego rdzenia (dwóch procesorów logicznych)
    3. dwa wątki do jednego rdzenia i dwa do innego rdzenia, do czterech różnych procesorów logicznych
    4. każdy wątek do innego rdzenia tego samego procesora
    5. każdy wątek do innego rdzenia, dwa do pierwszego procesora, dwa do drugiego

    [poprawność przypisania wątków można sprawdzać poprzez uruchomienie w osobnym terminalu narzędzia *htop -d 1* i sprawdzanie obciążenia odpowiednich procesorów logicznych / rdzeni (uwaga: *htop* numeruje procesory logiczne od 1)]

	czas	Gflops	przyspieszenie*	Ghz	flop/takt
wariant 1					
wariant 2					
wariant 3					
wariant 4					
wariant 5					

* przyspieszenie oznacza wzrost wydajności w stosunku do wersji jednowątkowej

4. Zadanie 3. Badanie wpływu przypisania wątków do rdzeni dla programu, którego czas wykonania zależny jest od czasów dostępu do pamięci DRAM komputera/serwera

1. Rozpakuj paczkę *latency_and_pinning.tgz*, skompiluj i uruchom w standardowej konfiguracji.
2. Dokonaj podobnej analizy jak dla zadania *latency_throughput_multithreaded_vector_flops,* tym razem dla problemu *random_pointer_chasing_mth* (w wersji wielowątkowej z paczki).
    1. Przeprowadź badania dla rozmiaru tablicy znalezionego w lab. 5 jako dającego reprezentatywne wyniki dla opóźnienia dostępu do pamięci DRAM (ustaw odpowiednio wartość parametru DEFAULT_WORKING_SIZE jako rozmiaru tablicy liczb podwójnej precyzji w pliku *random_pointer_chasing_mth.c* - rozmiar w bajtach uzyskiwany jest przez pomnożenie razy 8).

        [**Dla serwera Honorata wartości 64MB i wyższe dla rozmiaru tablicy w bajtach, przy skoku *stride* powyżej 64B (128B i więcej) powinny dać reprezentatywne wyniki – w eksperymentach można pozostawić domyślne wartości z paczki**]
    2. Liczbą wątków wykonujących obliczenia można sterować za pomocą parametru NO_TH w pliku r*andom_pointer_chasing_mth.c*.
        1. definicję w pliku można także zakomentować i przekazywać argument podczas kompilacji (np. *gcc -c -O3 -DNO_TH=2 ....*)
    3. Wyniki zapisz w tabeli, takiej jak w zadaniu 2, lecz zawierającej przepustowość w GB/s.

        (dodatkowo można uruchamiać obliczenia w ramach narzędzia *perf stat* i wykorzystując podaną częstotliwość pracy rdzenia obliczać wydajność pamięci (przepustowość) w B/takt – jest to dokładniejsze, ale także przybliżone, ze względu na uwzględnianie przez *perf stat* całego programu; w pełni precyzyjne obliczenia wydajności w flop/takt umożliwia zastosowanie interfejsu PAPI)

4. Wyciągnij wnioski z przeprowadzonych eksperymentów

Zauważ, że wydajności pozostają wciąż bardzo niskie w porównaniu z maksymalnymi przepustowościami uzyskanymi w laboratorium 3 programem *multiple_arrays* – wyciągnij wnioski, kiedy wielowątkowość może stać się istotnym mechanizmem zwiększania wydajności

Wyciągnij wnioski dotyczące przypisania wątków do rdzeni, porównując dwa wykorzystane dotychczas w laboratorium programy – jakie można zaobserwować różnice?

---------- 3.0 ----------

5. Skompiluj i uruchom zadanie z poprzedniego punktu z liczbą wątków równą 24,48 (pełne wykorzystanie SMT) i 96. Potwierdź obserwację dotyczącą typów obliczeń dla których, jak można wywnioskować z uzyskanych w tym i poprzednim punkcie wyników, szczególne przydatne może być wykorzystanie sprzętowej wielowątkowości?

   [w zadaniu 3 badany jest program, który nie wykorzystuje w znaczącym stopniu możliwości przetwarzania rdzeni procesorów, natomiast testuje układ pamięci – wątki generują żądania dostępu do pamięci, które są obsługiwane przez cały układ pamięci; na ile wielowątkowość i odpowiednie przypisanie wątków do rdzeni wpływają na wydajność w takim przypadku?, czy potwierdza to cechę działania układu pamięci polegającą na wzroście wydajności przy zwiększeniu liczby współbieżnie generowanych dostępów do pamięci?, na ile przekroczenie liczby procesorów logicznych przez liczbę wątków (sytuacja, kiedy na jednym rdzeniu pracuje więcej niż dwa wątki) może wpłynąć na wydajność?]

---------- 3.5 ----------

Dalsze kroki:

1. Przeprowadź badania dla algorytmu *random_pointer_chasing_mth* dla trzech dodatkowych rozmiarów tablicy znalezionych jako dające reprezentatywne wyniki dla opóźnienia dostępu do kolejnych poziomów pamięci podręcznej: L3, L2, L1 (sterując parametrem DEFAULT_WORKING_SIZE, należy także zmienić parametr DEFAULT_STRIDE na odpowiedni do pomiaru opóźnienia danego poziomu pamięci podręcznej). Wyniki wydajnościowe zapisz w trzech tabelach, takich jak w p.4.3. Wyciągnij wnioski z uzyskanych rezultatów.

   [szczególnie interesujące może być wykorzystanie pamięci L3 – współdzielonej przez wszystkie rdzenie pojedynczego mikroprocesora, można rozważyć np. dwa rozmiary tablicy o różnej proporcji do rozmiaru pamięci L3 (np. 4MB i 36MB, oba ze skokiem np. 64B) i zobaczyć jak wyglądać będzie sprawa w przypadku wielowątkowym przy różnych sposobach przypisania wątków do procesorów logicznych]

   [w przypadku pamięci L1 i L2 współdzielenie dotyczy tylko wątków pracujących na jednym rdzeniu przy wykorzystaniu simultaneous multithreading (hyperthreading)]
2. Przeprowadź eksperymenty dla innych wartości skoku, dla których mogą pojawić się ciekawe efekty, w przypadku pamięci każdego z rodzajów: DRAM, L3, L2, L1

---------- +0.5 za każdy poziom pamięci ----------

Sprawozdanie:

1. Zrealizowane kroki, spostrzeżenia z analizy kodu (a także ewentualnie odpowiadającego kodu asemblera i uzyskanych wartości liczników sprzętowych), wnioski itp. - zgodnie z regulaminem laboratoriów