



Szybki wstęp do OpenMP

czyli jak wycisnąć więcej z CPU i optymalizować równoległe obliczenia

Jan Iwaszkiewicz

Wydział Matematyki, Fizyki i Informatyki
Uniwersytet Gdański

30 kwietnia 2019

Plan prezentacji

- 1 Wstęp - czym jest OpenMP?
- 2 Teoria
 - Czy warto zrównoleglić obliczenia?
 - Praktyczne podejście
- 3 Pamięć
 - Ciągły dostęp do pamięci
 - Cache, cache i jeszcze raz cache
- 4 Synchronizacja i Pseudo-CRCW
 - Synchronizacja wątków
 - Pseudo-CRCW
- 5 Przykłady
 - Przykłady - prezentacja programów
 - Przykłady - optymalizacje kompilatora
- 6 Podsumowanie

Wstęp - czym jest OpenMP?

Wstęp - czym jest OpenMP?

OpenMP to interfejs umożliwiający tworzenie programów dla systemów wieloprocessorowych.

Dostępny dla języków C, C++ oraz Fortran.

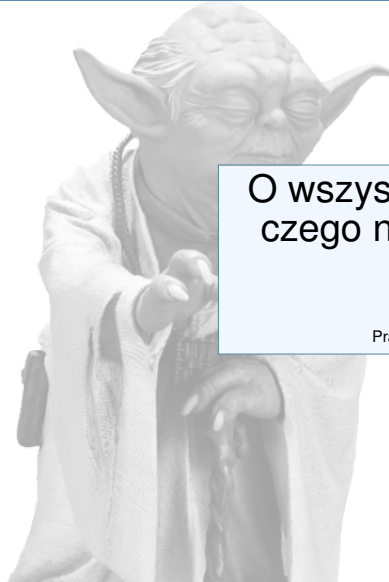
Dostosowany do wielu architektur, w tym też Unixowych. Składa się ze zbioru dyrektyw kompilatora, bibliotek i zmiennych środowiskowych.



Figure: Logo OpenMP, skrót HPC oznacza High-performance Computing.

Teoria

PARDOn't



O wszystkim zapomnij
czego nauczyłeś się!

Prawie wszystkim.

PARDOn't

Za

- Problem jest łatwy do zrównoleglenia
- Problem jest wystarczająco "duży"
- Dostęp do wyspecjalizowanego hardware (np. karty graficzne)

Przeciw

- "Skomplikowany" problem (np. brak możliwości ominięcia instrukcji warunkowych)
- Problem jest "mały" lub zbyt "prosty"
- Długi czas designu i optymalizacji takiego rozwiązania

Ograniczenia sprzętowe

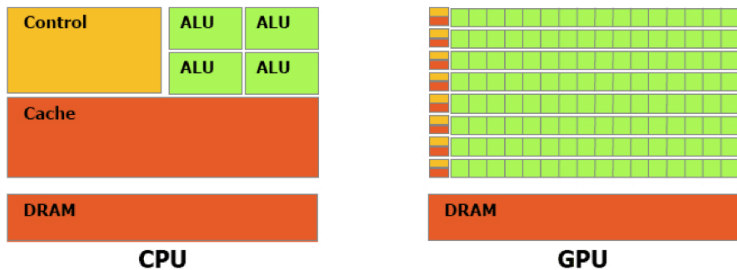


Figure: Porównanie budowy CPU i GPU.

"Z dedykacją..." - design algorytmów

Cel algorytmu, rozmiar i typ danych wejściowych, docelowy hardware, a nawet koszt zasilania sprzętu może mieć wpływ na wybrane rozwiązania przy tworzeniu kodu. Każdy nowy czynnik może drastycznie obniżyć jakość stworzonego programu.

Gdzie szukać inspiracji?

- LAPACK - Linear Algebra PACKage
- Intel[®] Math Kernel Library for Deep Neural Networks (Intel[®] MKL-DNN)
- Eigen

oraz wiele innych...

Pamięć

1 > 2 > 3 ... - zapis tablicy/wektora/tensora

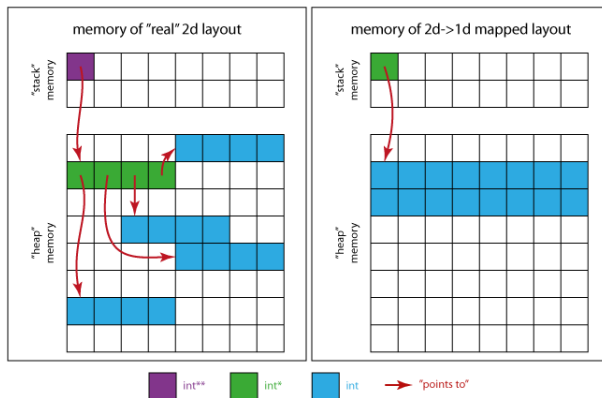


Figure: Zapis tablicy dwu-wymiarowej i jedno-wymiarowej w pamięci.

1 > 2 > 3 ... - jak iterować?

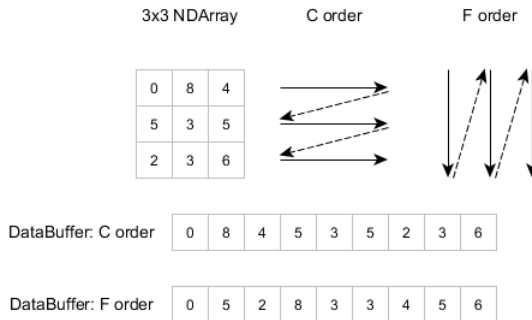
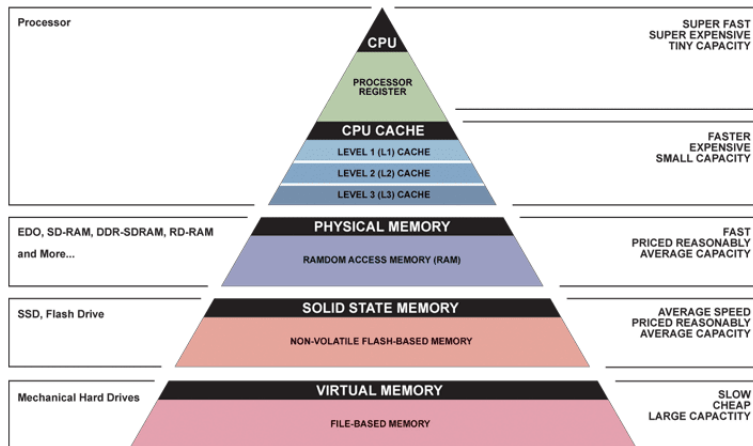


Figure: Zapisywanie pamięci rzędami oraz kolumnami w różnych językach programowania.

Droga od RAM do procesora



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Figure: Hierarchia pamięci.

Droga od RAM do procesora

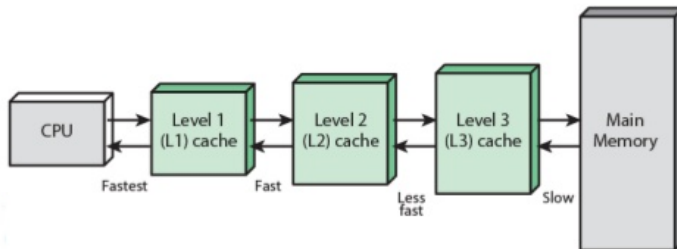


Figure: Droga jaką przebywają dane.

We wnętrzu CPU

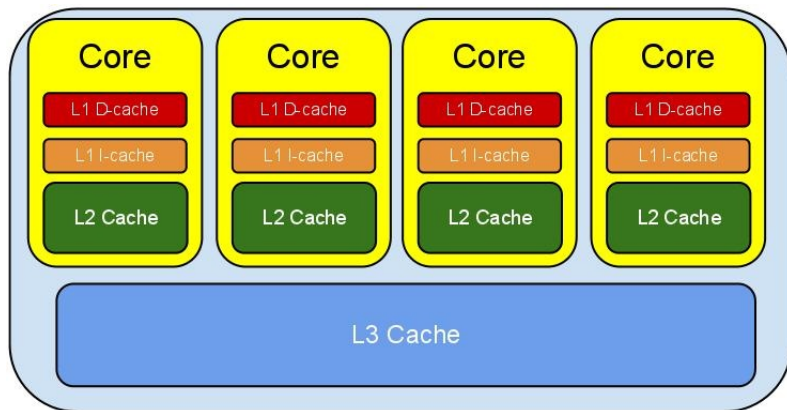


Figure: Uproszczony szkic podziału pamięci CPU.

Cache line

Cache line to najmniejszy blok danych transferowany i przechowywany w pamięci podręcznej. Żeby dowiedzieć się jaki rozmiar linii jest przyjęty w danym procesorze, pod Linuxem, wystarczy wywołać komendę:

```
cat /sys/devices/system/cpu/cpu0/cache/index0/coherency_line_size
```

Najbardziej typowy rozmiar to **64B**, jednak zdarzają się też rozmiary od 32B do 256B.

False sharing

False sharing niepożądane zjawisko podczas odczytu i zapisu danych przez różne wątki. **Wątki pracujące na tym samym cache line mogą doprowadzić do stalling'u** (zastoju w transferze danych) poprzez ciągłe żądania aktualizacji danych.

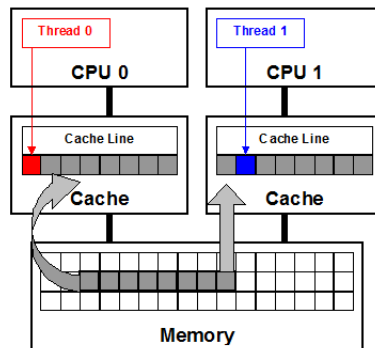


Figure: Przykład momentu, w którym zachodzi false sharing.

Garbage collector

Dlaczego większość wydajnych aplikacji wielowątkowych oraz matematycznych nie korzysta z "nowych" języków programowania? Jednym z minusów jest garbage collector.

Problemy

- Możliwość zatrzymania całego programu na czas działania - brak użycia pożądanego ciągłego wielowątkowości
- Koszt pracy garbage collector'a samego w sobie (szczególnie dawniej) - proces zajmuje cenny dla HPC czas
- Zdarzają się zmiany w adresowaniu - przemieszczanie danych

Synchronizacja i Pseudo-CRCW

Bariery

```
#pragma omp barrier
```

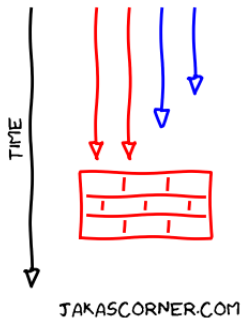


Figure: Szkic działania bariery, czerwone wątki natrafiły już na blokadę, niebieskie dalej pracują.

Odczyt danych - współdzielona pamięć

```
#pragma omp ... shared(a, b, c)
```

Podczas odczytu nie wystąpią żadne konflikty.

Zapis danych - operacje atomiczne

```
#pragma omp atomic
```

```
#pragma omp critical
```

Podczas zapisu nie wystąpią żadne konflikty. Jednak takie rozwiązanie prowadzi to do **serializacji** zapisów i znaczącego spowolnienia działania programu.

Przykłady

Przykłady - programy

Przykładowe operacje

- Dodawanie dwóch wektorów - $O(n)$
- Iloczyn skalarny dwóch wektorów - $O(n)$
- Mnożenie dwóch kwadratowych macierzy - $O(n^3)$

Spójrzmy jak działają przygotowane przykłady i omówmy ich działanie.

Flagi kompilatora

Spróbujmy zoptymalizować kod przy pomocy kompilatora, użyjmy do tego flag:

```
-O3 -funroll-loops
```

[Link do opisu flag dla kompilatora GCC](#)

Podsumowanie

Podsumowanie

Dziękuję za uwagę!
Zapraszam do dyskusji oraz pytań.