Paralelní a distribuované výpočty (B4B36PDV)

Jakub Mareček, Michal Jakob

jakub.marecek@fel.cvut.cz

Artificial Intelligence Center
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Paralelní a distribuované výpočty

Paralelní programování

- 1 stroj, vícero jader, hierarchie vyrovnávacích pamětí, sdílená paměť,
- 1 program, použití synchronizačních primitiv, datové struktury
- OpenMP, C++20, Go, Rust

Programování v distribuovaných systémech

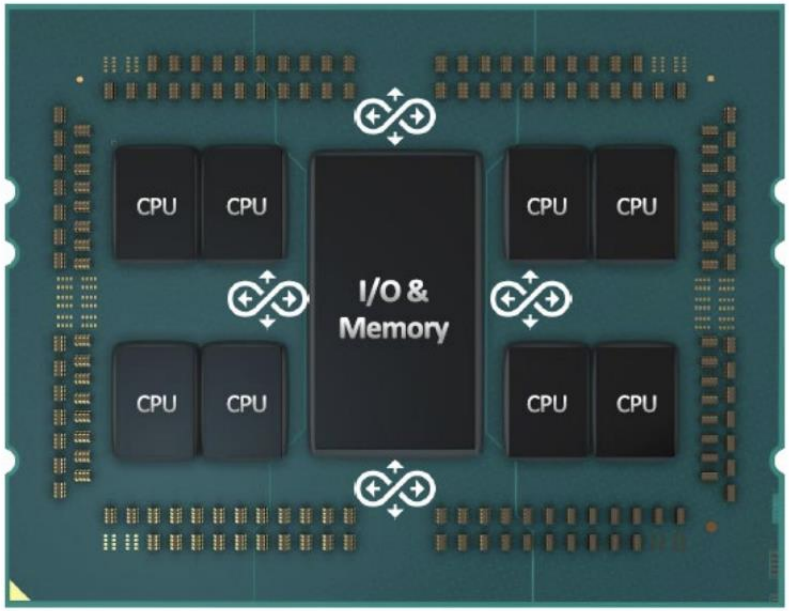
- vícero strojů komunikujících po síťových rozhraních (např. InfiniBand)
- distribuovaná data
- komunikační složitost, „HW na zakázku”
- MPI, k8s, Spark

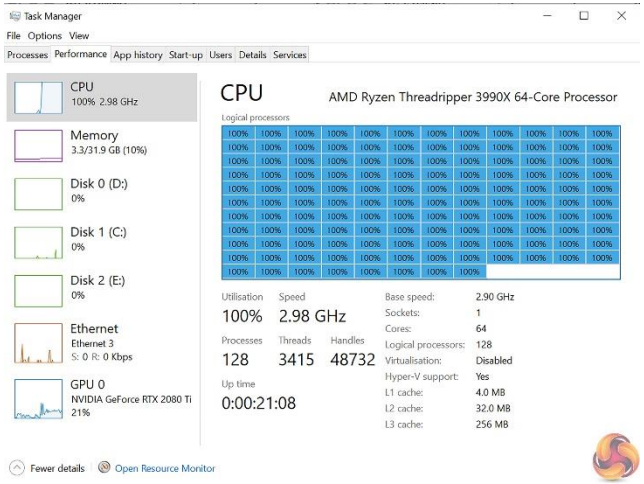
Rychleji nalézt řešení

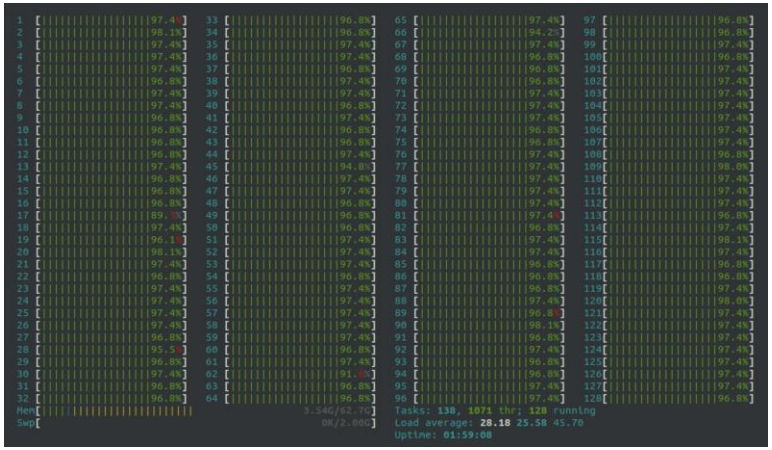
\+ Zvýšit robustnost řešení

Motivace

Co je to moderní procesor?





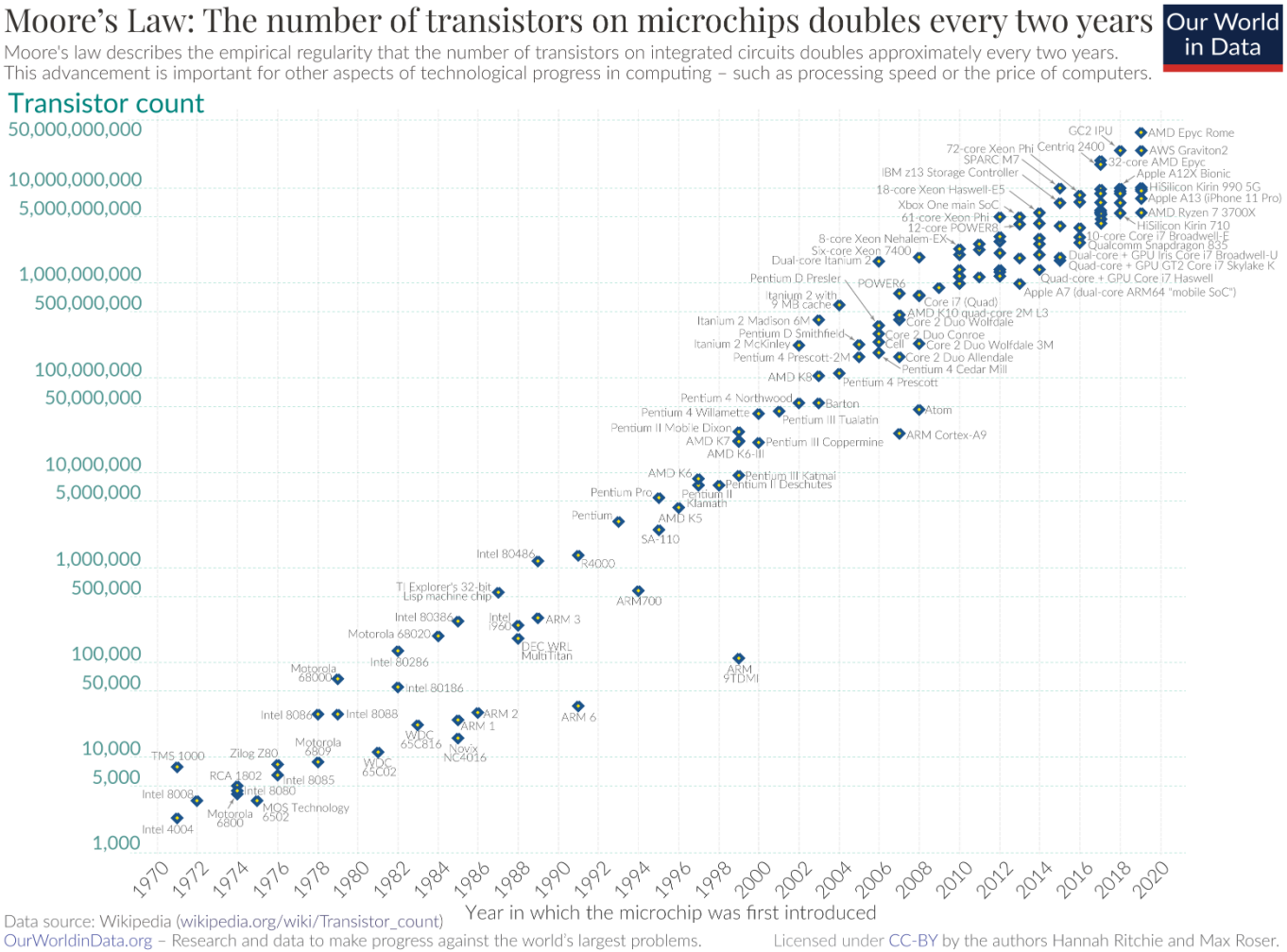


Obrázky převzaty z:

- https://www.tomshardware.com/news/amd-zen-3-ryzen-5000-release-date-specifications-pricing-benchmarks-all-we-know
- https://www.amd.com/system/files/documents/tr-pro-thought-leadership.pdf
- https://techgage.com/article/amd-ryzen-threadripper-3990x-64-core-linux-performance/

Motivace:

Nárůst počtu transistorů

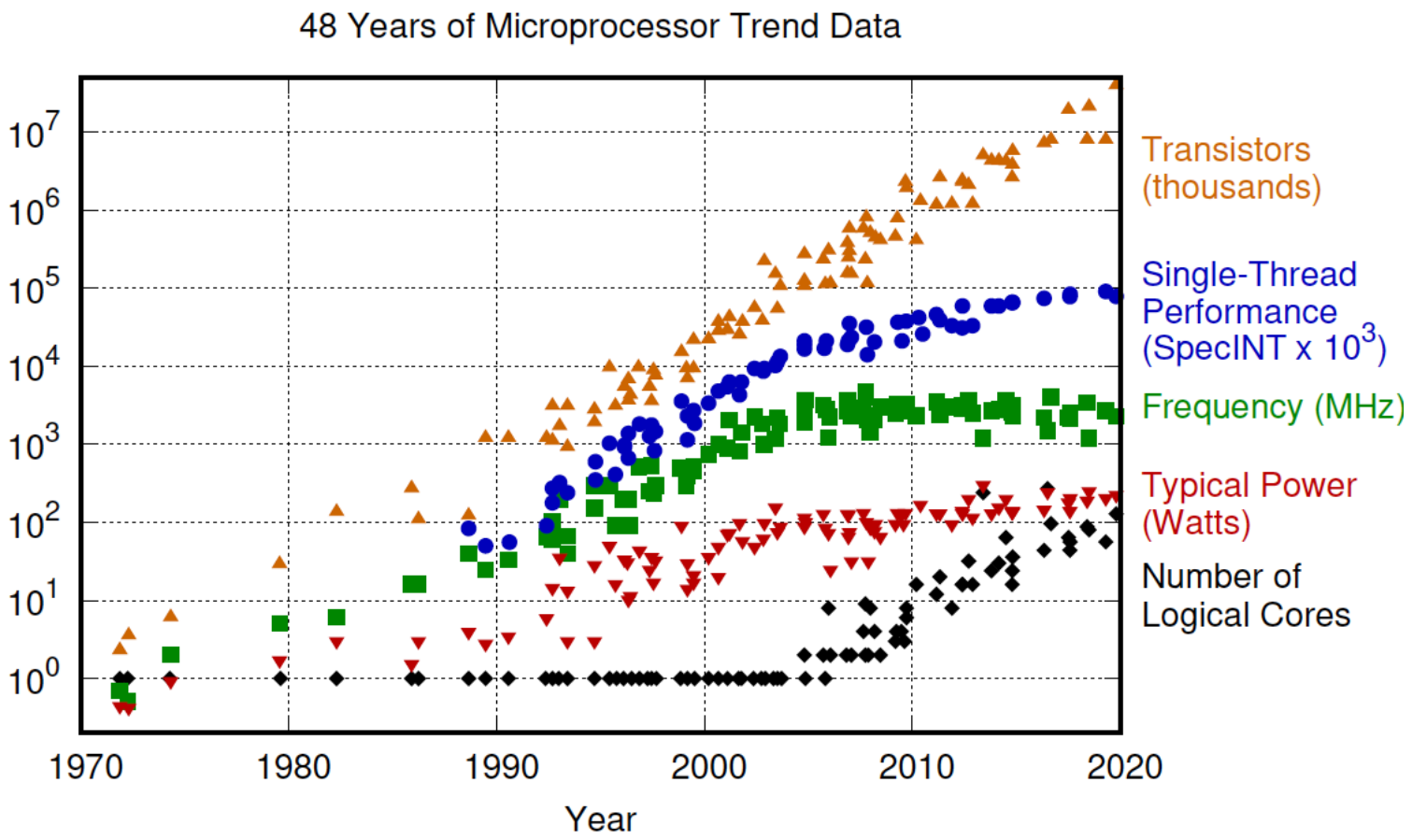


Obrázky převzaty z:

- https://en.wikipedia.org/wiki/Moore%27s_law

Motivace

Nárůst výkonu jednotlivého jádra; počtu jader

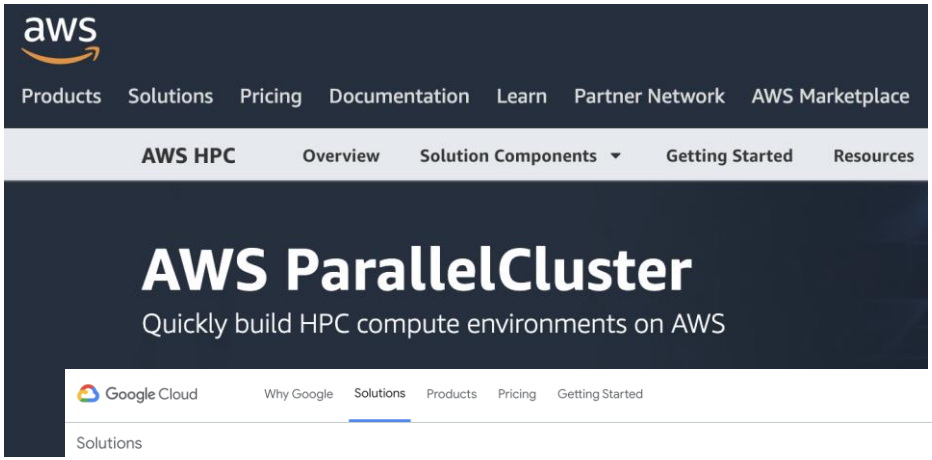


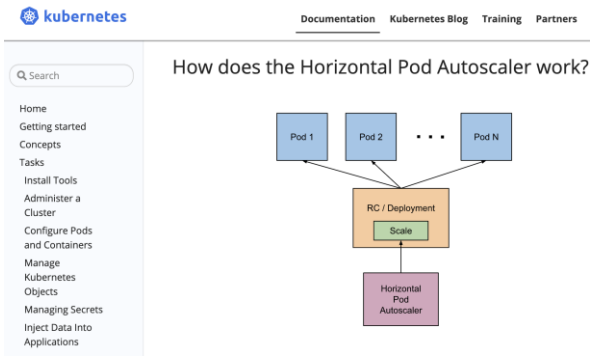
Data viz:

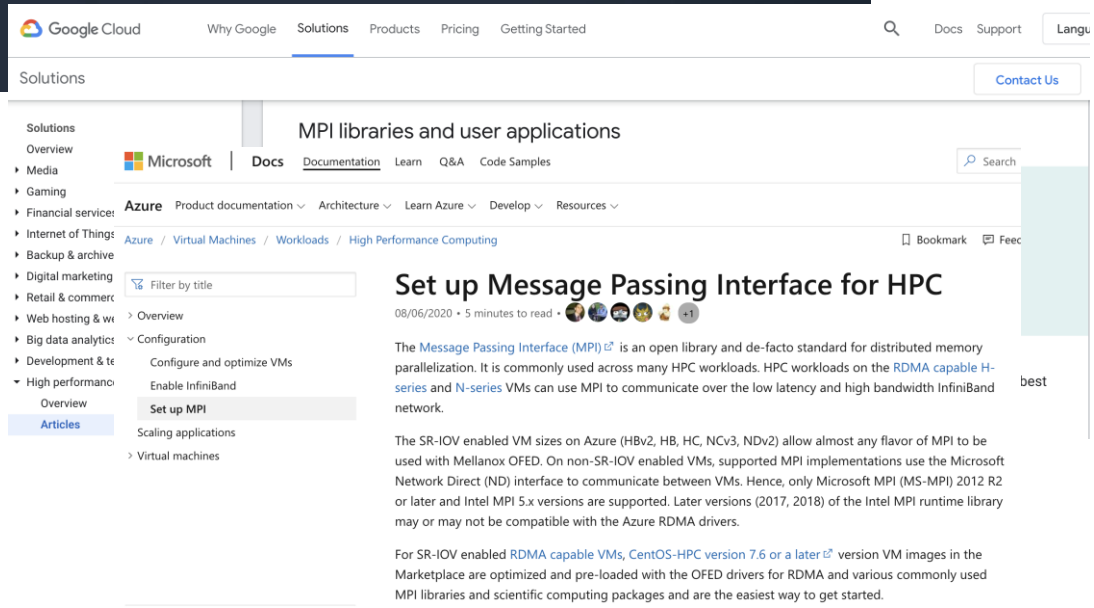
- https://zenodo.org/record/3947824#.YCBhyhNKhpI

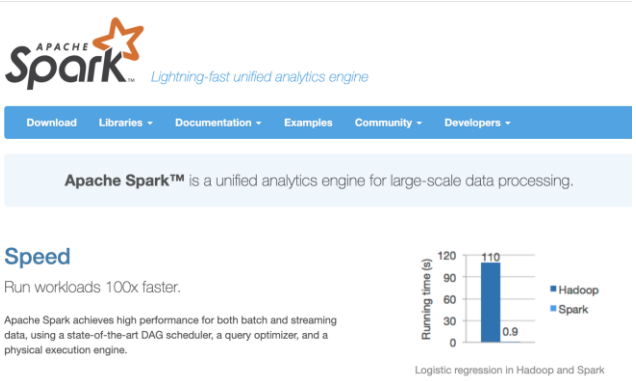
Motivace

Máme cloud, ne?









Více viz:

- https://docs.aws.amazon.com/parallelcluster/latest/ug/tutorials_03_batch_mpi.html
- https://cloud.google.com/solutions/best-practices-for-using-mpi-on-compute-engine
- https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/hpc/setup-mpi
- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
- https://spark.apache.org/

Kdo jsme

Přednášející

Jakub Mareček

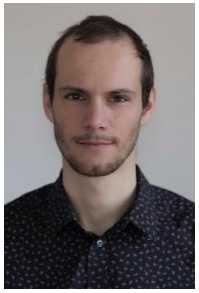
Michal Jakob

Cvičící

Peter Macejko

David Fiedler

Jan Mrkos



David Milec

Zd. Rozsypálek

Tom Rouček

Organizace a přehled

- Paralelní část
  - Získat základní informace a prostor pro praktické zkušenosti v oblasti programování efektivních paralelních programů
  - Paralelní programování jednoduchých algoritmů
  - Vliv různých způsobů paralelizace na rychlost výpočtu
- Distribuovaná část
  - Problémy v distribuovaných systémech (shoda, konzistence dat)
  - Navržení robustných řešení

- CourseWare
  - https://cw.fel.cvut.cz/wiki/courses/b4b36pdv/start
- Chcete kvízy?

Přehled paralelní části

- Základní úvod
  - Vlákna, synchronizace, mutexy
  - Pthread (již by jste měli znát), C++11 thready, C++20 jthready
- OpenMP
  - Specifikace nadstavby nad kompilátorem (např. C, C++, Fortran) a knihovny pro zjednodušení implementace paralelních programů
- Techniky dekompozice
- Datové struktury umožňující přístup vícero vláken
- Základní paralelní řadící algoritmy a vektorové instrukce
- Základní paralelní maticové algoritmy

Materiály k paralelní části

- Standardní učebnice: The Art of Multiprocessor Programming (by Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear). Druhé vydání (září 2020). Viz také https://www.youtube.com/watch?v=nrUszqrlvi8.
- Standardní dokumentace OpenMP: https://www.openmp.org/resources/refguides/ je povolena na praktické zkoušce. Nikoli příklady https://github.com/OpenMP/Examples/
- Velmi praktické rady: Using OpenMP (Portable Shared Memory Parallel Programming, by Barbara Chapman, Gabriele Jost and Ruud van der Pas). Vydání z roku 2007 je dostupné přes NTK.
- Neformální úvod: Programming on Parallel Machines (by Norm Matloff), 2012, k dispozici zdarma on-line

Hodnocení

- Domácí úkoly (40%)
  - Malé domácí úkoly (7x)
  - Velké domácí úkoly (2x)
- Praktický test z paralelního programování (20%).
  Loni online, letos nejspíše zpět v učebně.
- Teoretický test (40%).
  Loni poprvé v Brute jako výběr z vice možností.

Pro úspěšné ukončení musíte získat alespoň 50% z každé části

Hodnocení

- Malá tolerance zpoždění u odevzdávání úloh
- Problémy je dobré řešit včas na fóru CourseWare (nikoli však nutně ve vláknu, které se posílá všem)
- Prostředí BRUTE, CLion (https://download.cvut.cz/jetbrains/), příp. Windows Subsystem for Linux

Malé úlohy

Zpoždění	Penalizace
$0h < x \leq 1h$	-0.5b
$1h < x \leq 24h$	-1b
$24h < x$	-2b

Semestrální úlohy

Zpoždění	Penalizace
$0h < x \leq 1h$	-1b
$1h < x \leq 12h$	-2b
$12h < x \leq 3d$	-6b
$3d < x \leq 6d$	-10b
$6d < x$	-100%

Co udělat pro úspěšné zvládnutí PDV?

- Programovat
  - zkoušejte si kódy z přednášek, upravujte jej, analyzujte co se stane
  - nechte si čas na vypracování domácích úkolů

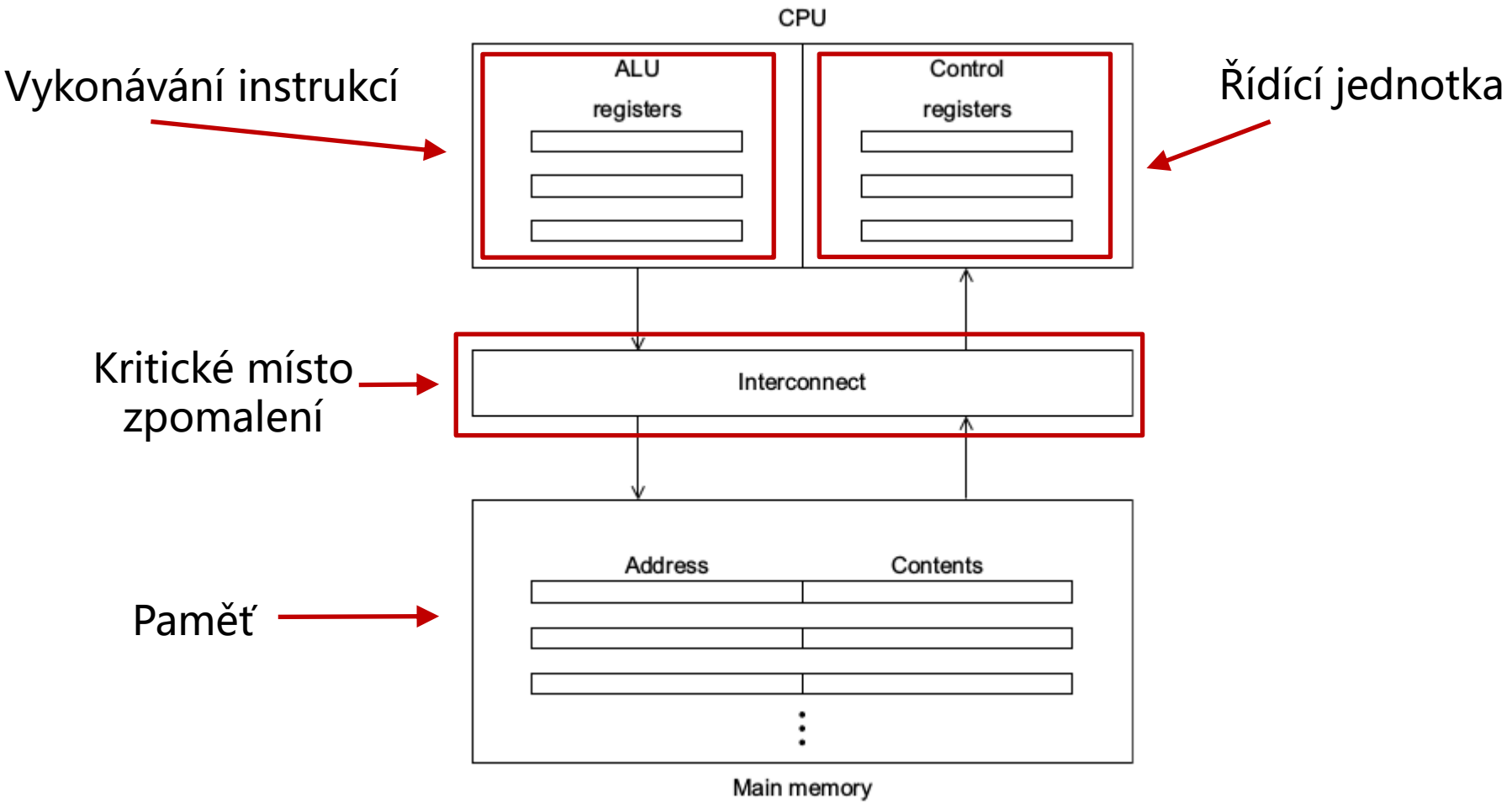
- Přemýšlet
  - paralelní / distribuované programy se špatně ladí
  - chyby ve vícevláknové aplikaci v debug-módu neodhalíte (mohou pomoci ladící výpisy)
  - pokud program nepracuje jak očekáváte (např. není dostatečně rychlý, výsledek není správný), **zastavte se a zamyslete se proč tomu tak je**

Přehled dnešní přednášky

- Potřebný HW základ a krátká historie
- Jednoduché příklady (jak to dělat a nedělat);
  Vliv architektury
- Pthreads vs. C++ vs. OpenMP
- Nové státnicové otázky

Potřebný HW základ

Von Neumannova architektura



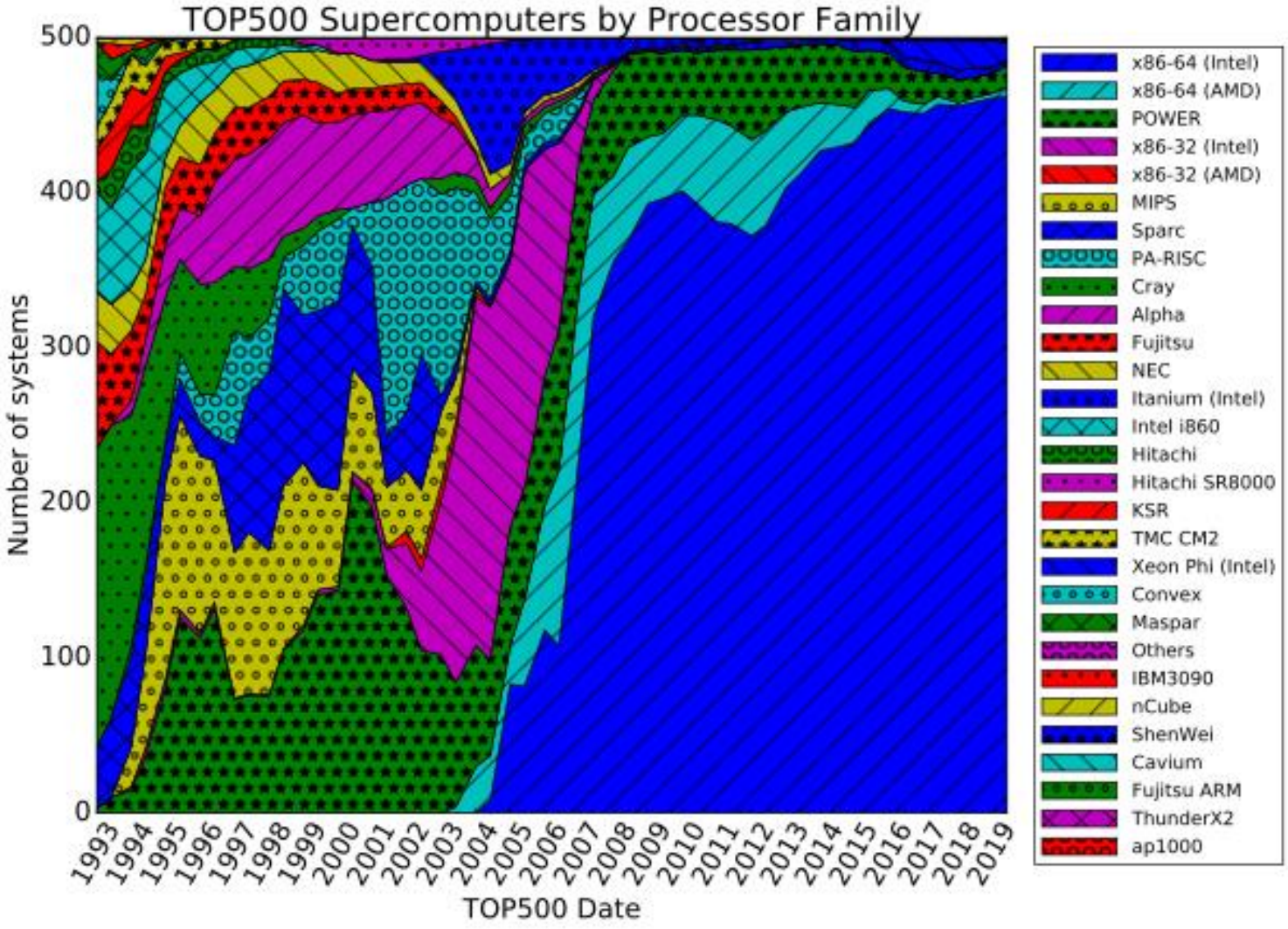
Obrázek převzat z knihy Parallel Programming (by Peter Pacheco)

Krátká historie paralelních výpočtů

Dnešní paralelní stroje

Krátká historie paralelních výpočtů

TOP 500 superpočítačů



Zdroj: https://en.wikipedia.org/wiki/TOP500

Krátká historie paralelních výpočtů

Co u nás?

- RCI ČVUT cluster
  - n01-20 CPU nodes: 24 cores/48 threads 3.2GHz (2 x Intel Xeon Scalable Gold 6146), 384GB RAM,
  - n21-n32 GPU nodes: 36 cores/72 threads 2.7GHz (2 x Intel Xeon Scalable Gold 6150), 384GB RAM, 4 x Tesla V100 with NVLink,
  - n33 multi-CPU node: 192 cores/ 384 threads 2.1GHz (8 x Intel Xeon Scalable Platinum 8160), 1536GB RAM

Krátká historie paralelních výpočtů

Co u nás?

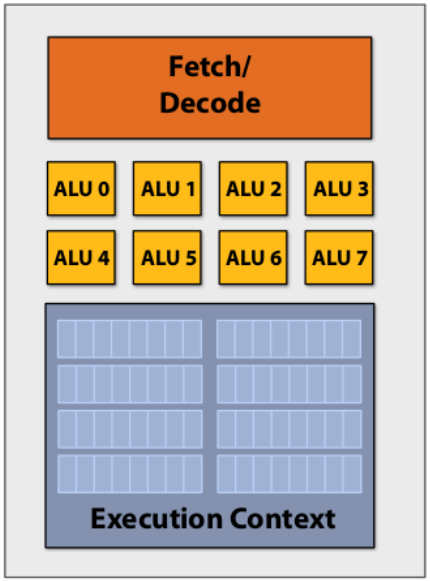
- Metacentrum
  - spojení výpočetních prostředků akademické sítě
  - volně dostupné pro akademické pracovníky, studenty
  - mnoho dostupných strojů (CPU, GPU, Xeon Phi)
  - https://metavo.metacentrum.cz/pbsmon2/hardware

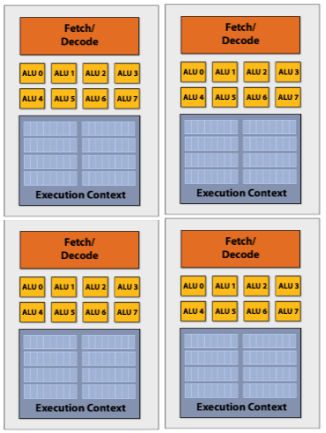
- IT4Innovations (www.it4i.cz)
  - 30 000 jader v Ostravě, příkon přes 1MW
  - komerční výpočty, lze zažádat a získat výpočetní čas pro výzkum

Potřebný HW základ

Abstrakce paralelního hardware dle Flynnovy taxonomie

- SIMD (Single Instruction Multiple Data)
  - Jedna řídící jednotka, vícero ALU jednotek
  - Datový paralelismus
  - Vektorové procesory, GPU
  - Běžné jádra CPU podporují SIMD paralelizmus
    - instrukce SSE, AVX
- MIMD (Multiple Instruction Multiple Data)
  - Více-jádrové procesory
    - např. i v mobilních telefonech
  - Různé jádra vykonávají různé instrukce
  - Víceprocesorové počítače



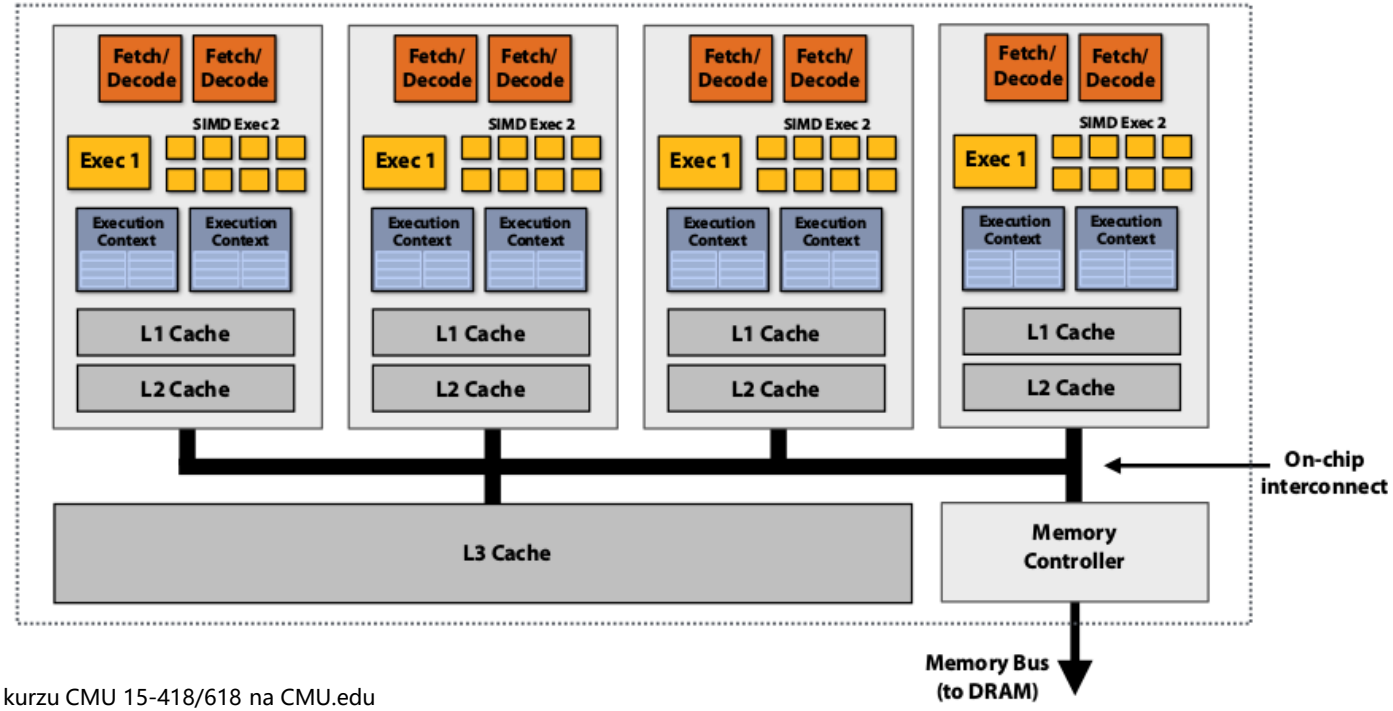


Obrázky převzaty z kurzu CMU 15-418/618 na CMU.edu

Potřebný HW základ

Abstrakce levného moderního procesoru

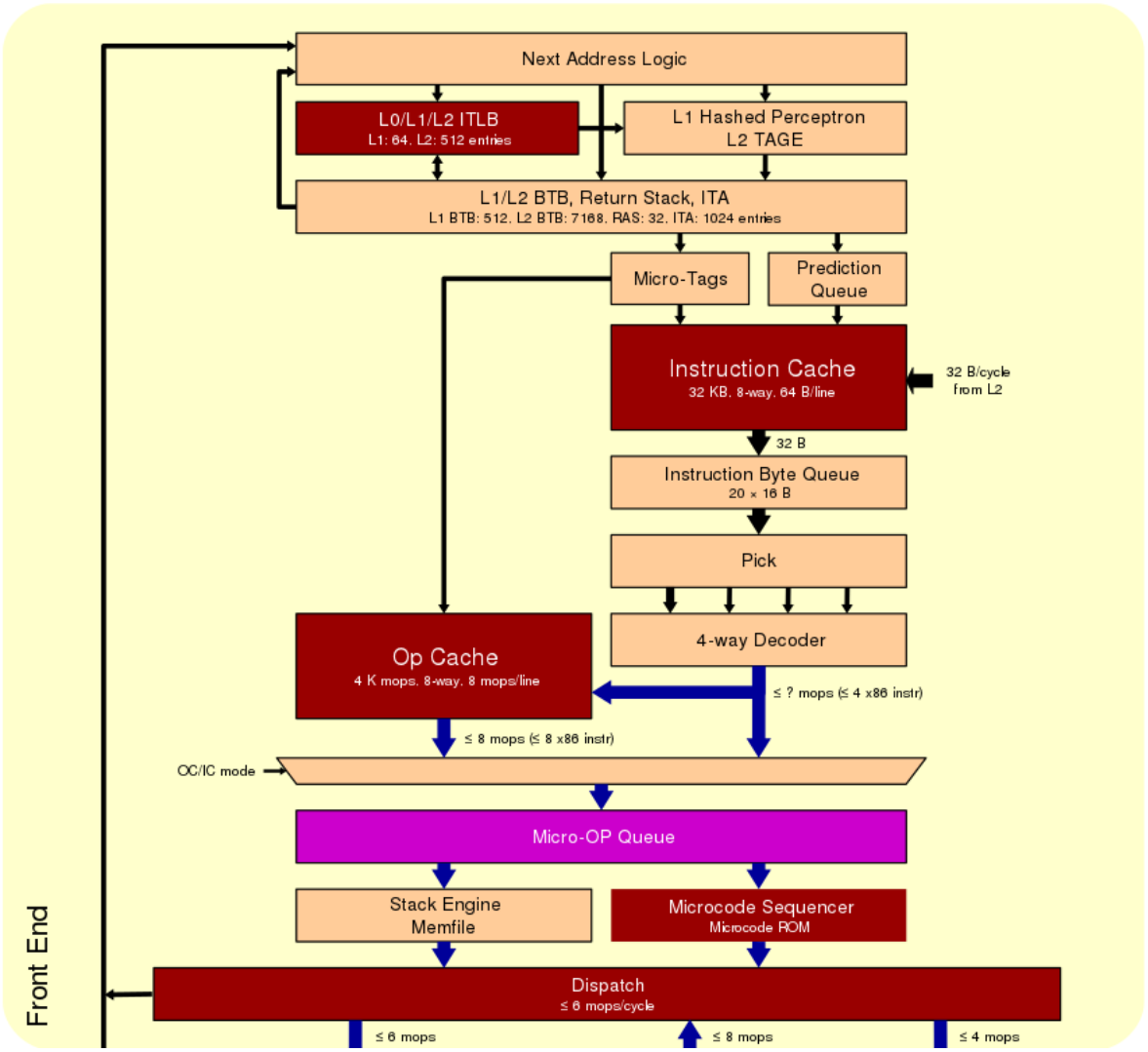
- Vyrovnávací paměť (CPU cache)
  - Programy často přistupují k paměti lokálně (lokalita v prostoru a čase)
  - Cache se upravuje po řádcích (lines)
- Každé jádro má vlastní cache + existuje společná cache



Obrázek převzaty z kurzu CMU 15-418/618 na CMU.edu

Potřebný HW základ

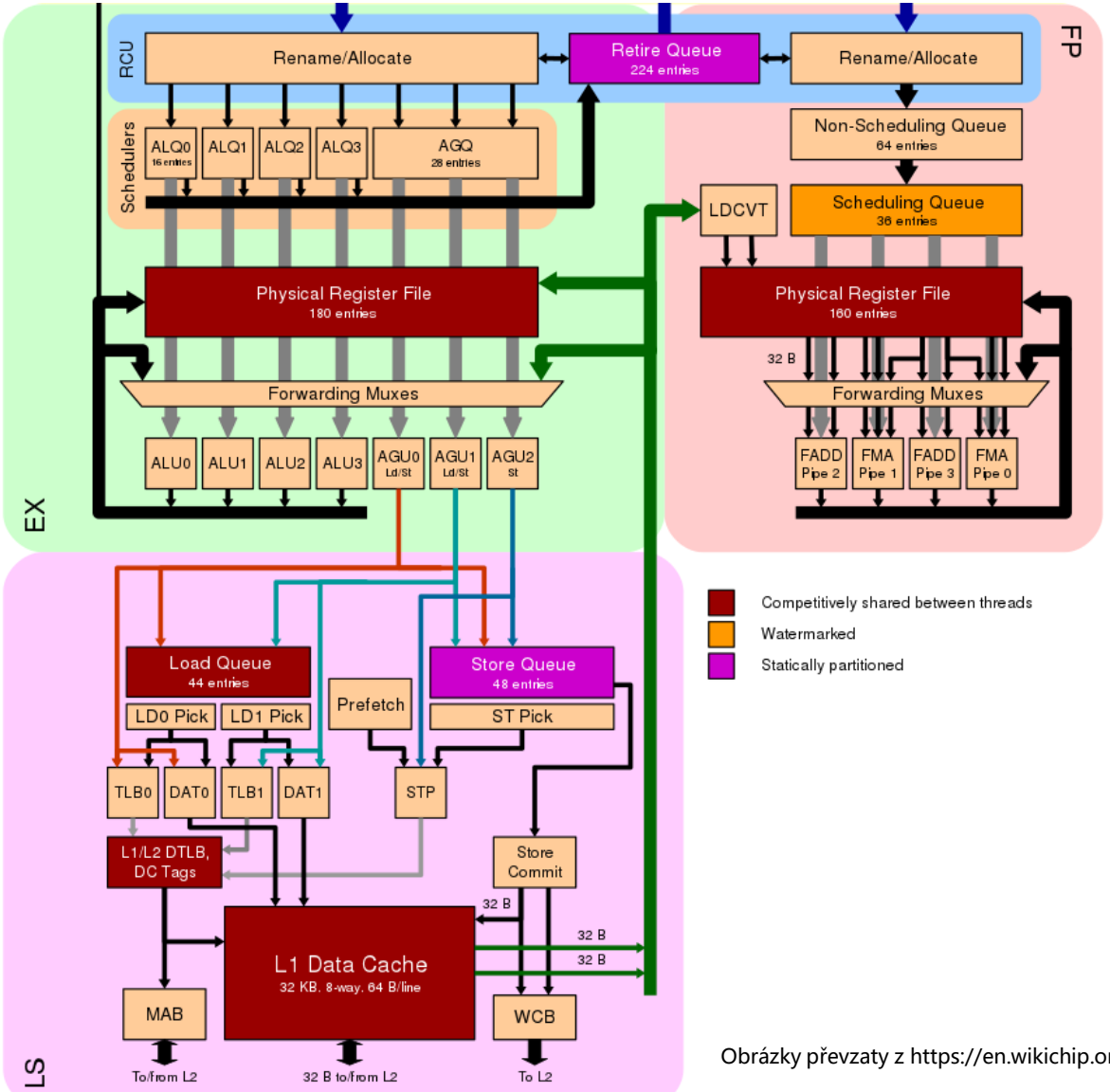
Realita levného moderního procesoru je podstatně složitější



Obrázky převzaty z https://en.wikichip.org/wiki/amd/microarchitectures/zen_3

Potřebný HW základ

Realita levného moderního procesoru je podstatně složitější



Obrázky převzaty z https://en.wikichip.org/wiki/amd/microarchitectures/zen_3

Potřebný HW základ

Pipelines

- Zopakujeme 5 konceptů z APO.
- Paralelizace na úrovni instrukcí (ILP). Příklad:
  - Chceme sečíst 2 vektory reálných čísel (float [1000])
  - 1 součet – 7 operací
    - Načtení (fetch)
    - Porovnání exponentů
    - Posun
    - Součet
    - Normalizace
    - Zaokrouhlení
    - Uložení výsledku
  - Bez ILP – 7x1000x (čas 1 operace; 1ns)

Potřebný HW základ

Pipelines

- Paralelizace na úrovni instrukcí (ILP)
- Příklad:
  - Chceme sečíst 2 vektory reálných čísel (float [1000])
  - 1 součet – 7 operací
  - Bez ILP – 7x1000x (čas 1 operace; 1ns)
  - ILP (a 7 jednotek) – 1005 ns

Table 2.3 Pipelined Addition. Numbers in the Table Are Subscripts of Operands/Results

Time	Fetch	Compare	Shift	Add	Normalize	Round	Store
0	0						
1	1	0					
2	2	1	0				
3	3	2	1	0			
4	4	3	2	1	0		
5	5	4	3	2	1	0	
6	6	5	4	3	2	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
999	999	998	997	996	995	994	993
1000		999	998	997	996	995	994
1001			999	998	997	996	995
1002				999	998	997	996
1003					999	998	997
1004						999	998
1005							999

Obrázek převzat z knihy Parallel Programming (by Peter Pacheco)

Potřebný HW základ

Superskalární procesory

- Současné vyhodnocení vícero instrukcí
  - uvažme cyklus

```
for (i=0; i<1000; i++)
    z[i]=x[i]+y[i];
```

  - jedna jednotka může počítat z[0], druhá z[1], …
- Spekulativní vyhodnocení

```
z = x + y;
if (z > 0)
    w = x;
else
    w = y;
```

https://cw.fel.cvut.cz/wiki/_media/courses/b35apo/en/lectures/06/b35apo_lecture06-speculative.pdf

Vliv architektury

Cache

- Proč je důležité vědět o architektuře?
  - Uvažme příklad násobení matice vektorem

```
int x[MAXIMUM], int y[MAXIMUM], int A[MAXIMUM*MAXIMUM]
```

Varianta A

```
for ( int i = 0; i < MAXIMUM ; i ++)
    for ( int j = 0; j < MAXIMUM ; j ++)
        y[i] += A->at(i * MAXIMUM + j)*x[j];
```

Varianta B

```
for ( int j = 0; j < MAXIMUM ; j ++)
    for ( int i = 0; i < MAXIMUM ; i ++)
        y [i] += A->at(i * MAXIMUM + j)*x[j];
```

Který kód bude rychlejší?

Vliv architektury

Cache

```
for ( int i = 0; i < MAXIMUM ; i ++)
    for ( int j = 0; j < MAXIMUM ; j ++)
        y[i] += A->at(i * MAXIMUM + j)*x[j];
```

```
for ( int j = 0; j < MAXIMUM ; j ++)
    for ( int i = 0; i < MAXIMUM ; i ++)
        y [i] += A->at(i * MAXIMUM + j)*x[j];
```

- Pole jsou v paměti uložena sekvenčně (po řádcích)
- CPU při přístupu k A[0][0] načte do cache vícero hodnot (cache line)

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

- Při přístupu k A[1][0] se změní celý řádek

V rámci paralelních programů může k podobným problémům docházet častěji

Paralelizace

Jednoduchý příklad

- Suma vektoru čísel

0	1	2	3	4	5	6	…	…	$5x10^9$
17	2	9	4	22	0	1			8

Jak paralelizovat?

- Mějme 4 jádra – každé jádro může sečíst čtvrtinu vektoru, pak sečteme částečné součty

Vláken	1	2	3	4
Čas	0.389s	0.262s	0.258s	0.244s

Měřeno na 8 jádrovém (16 vláknovém) Intel Xeon v4 E5-2620

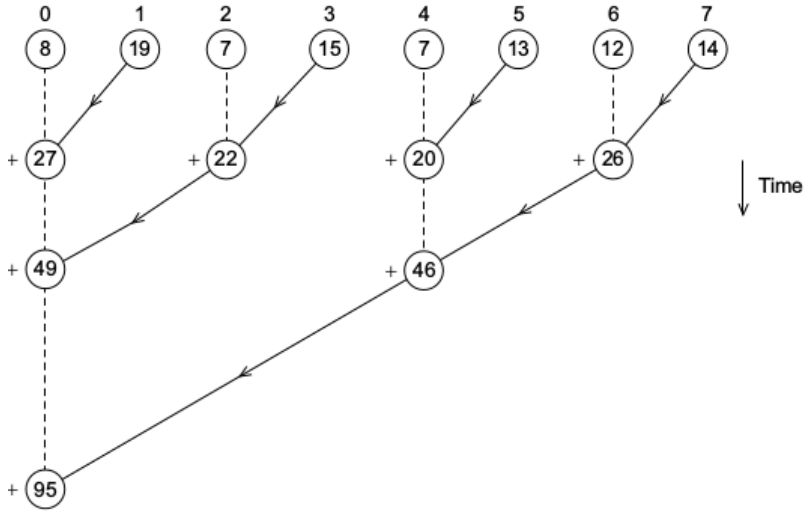
Paralelizace

Jednoduchý příklad

- Suma vektoru čísel

Co když máme tisíce jader?

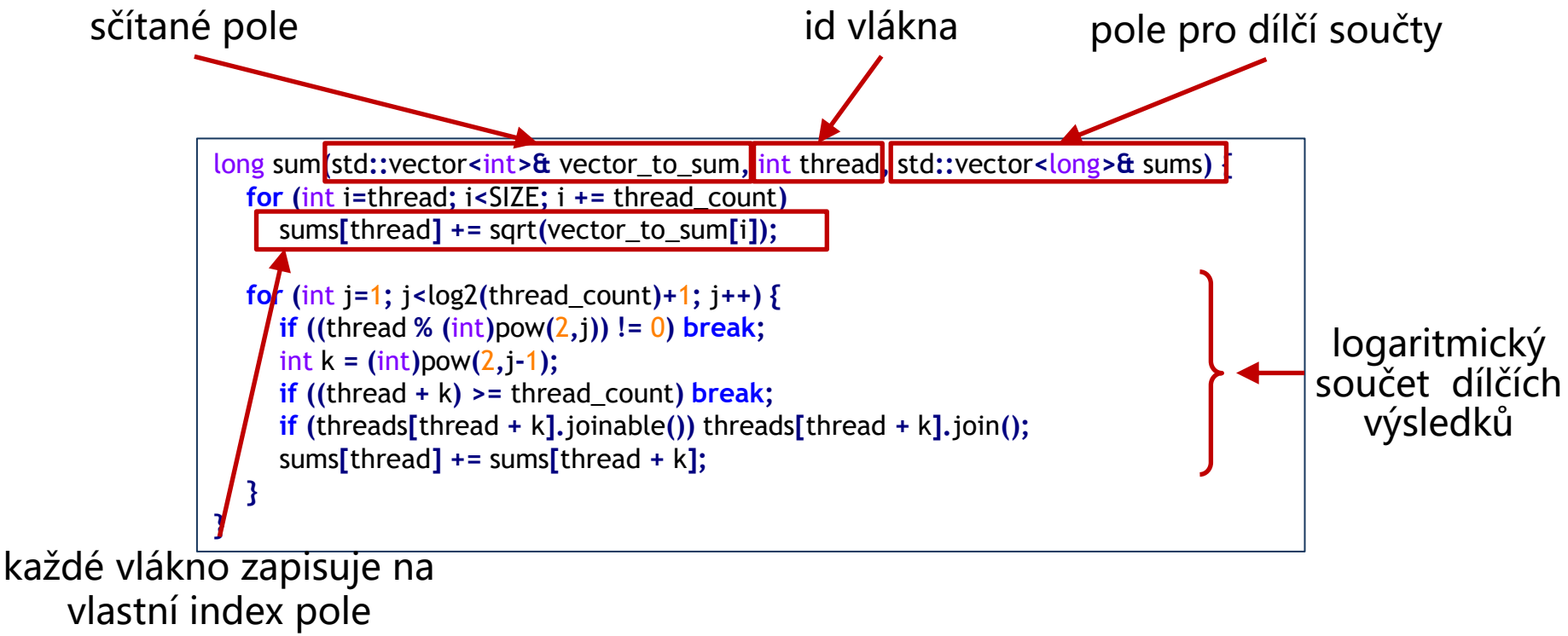
- Pokud částečné součty sčítá pouze jedno jádro, kód není velmi efektivní



Obrázek převzat z knihy Parallel Programming (by Peter Pacheco)

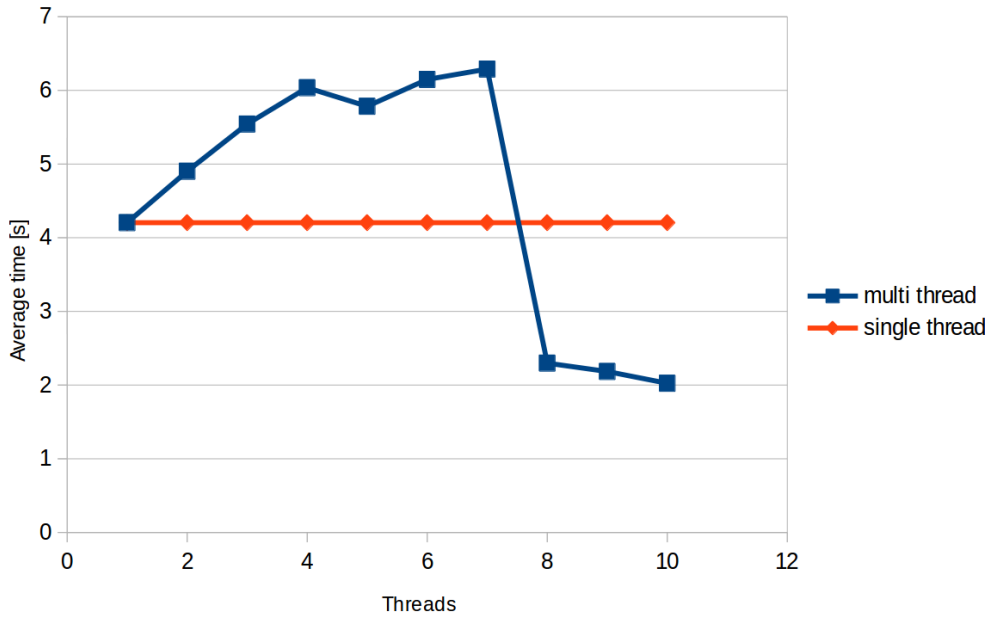
Pokročilejší příklad

- Zkusme sčítat celou část druhých odmocnin



Pokročilejší příklad

Jak nám to bude fungovat?



Nic moc :(

Měřeno na 8 jádrovém (16 vláknovém) Intel Xeon v4 E5-2620

Pokročilejší příklad

Kde je chyba?

```
long sum(std::vector<int>& vector_to_sum, int thread, std::vector<long>& sums) {
    for (int i=thread; i<SIZE; i += thread_count)
        sums[thread] += sqrt(vector_to_sum[i]);

    for (int j=1; j<log2(thread_count)+1; j++) {
        if ((thread % (int)pow(2,j)) != 0) break;
        int k = (int)pow(2,j-1);
        if ((thread + k) >= thread_count) break;
        if (threads[thread + k].joinable()) threads[thread + k].join();
        sums[thread] += sums[thread + k];
    }
}
```

každé vlákno zapisuje na vlastní index pole

0	1	2	3	4	5	6	7	8	9
17	2	9	4	22	0	1	0	0	8

Pokročilejší příklad

Kde je chyba?

```
long sum(std::vector<int>& vector_to_sum, int thread, std::vector<long>& sums) {
    for (int i=thread; i<SIZE; i += thread_count)
        sums[thread] += sqrt(vector_to_sum[i]);

    for (int j=1; j<log2(thread_count)+1; j++) {
        if ((thread % (int)pow(2,j)) != 0) break;
        int k = (int)pow(2,j-1);
        if ((thread + k) >= thread_count) break;
        if (threads[thread + k].joinable()) threads[thread + k].join();
        sums[thread] += sums[thread + k];
    }
}
```

- vlákno 0 upraví hodnotu
- jenže vlákno 0 má celý vektor **sums** v cache jádra
- a podobně i jiné vlákna
- při změně 1 hodnoty se musí zabezpečit konzistence

0	1	2	3	4	5	6	7	8	9
17	2	9	4	22	0	1	0	0	8

False Sharing

False Sharing

možné řešení

```
long sum_local(std::vector<int>& vector_to_sum, int thread, std::vector<long>& sums) {
    long local = 0;
    for (int i=thread; i<SIZE; i += thread_count) {
        local += sqrt(vector_to_sum[i]);
    }
    sums[thread] = local;

    for (int j=1; j<log2(thread_count)+1; j++) {
        if ((thread % (int)pow(2,j)) != 0) break;
        int k = (int)pow(2,j-1);
        if ((thread + k) >= thread_count) break;
        if (threads[thread + k].joinable()) threads[thread + k].join();
        local += sums[thread + k];
    }
    sums[thread] = local;
}
```

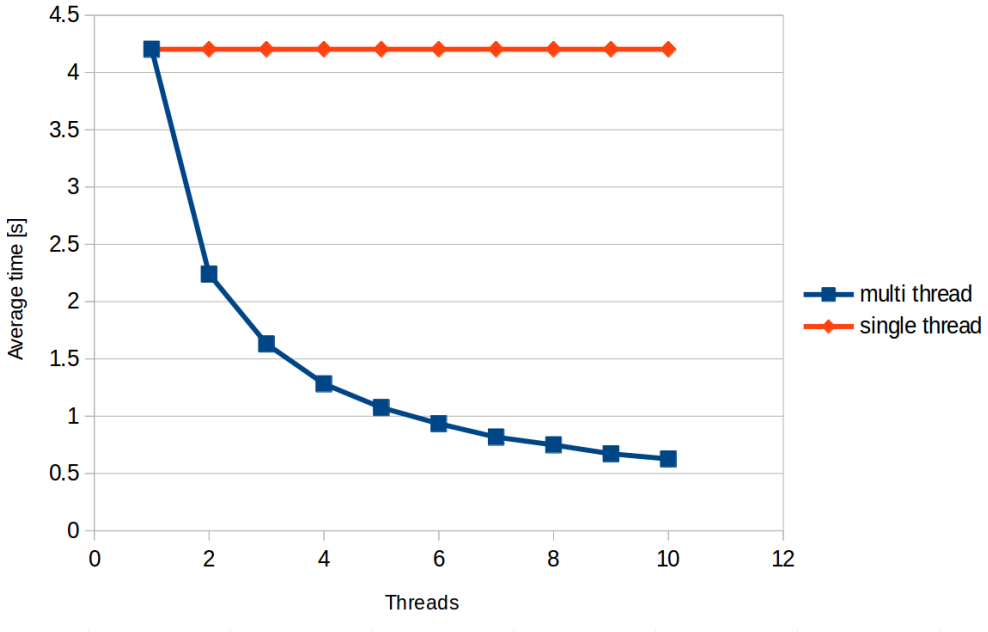
každé vlákno zapisuje do lokální proměnné

pouze finální výsledek se zapíše do vektoru

Potřebný HW základ

False Sharing

lokální proměnná – opravdu to pomůže?



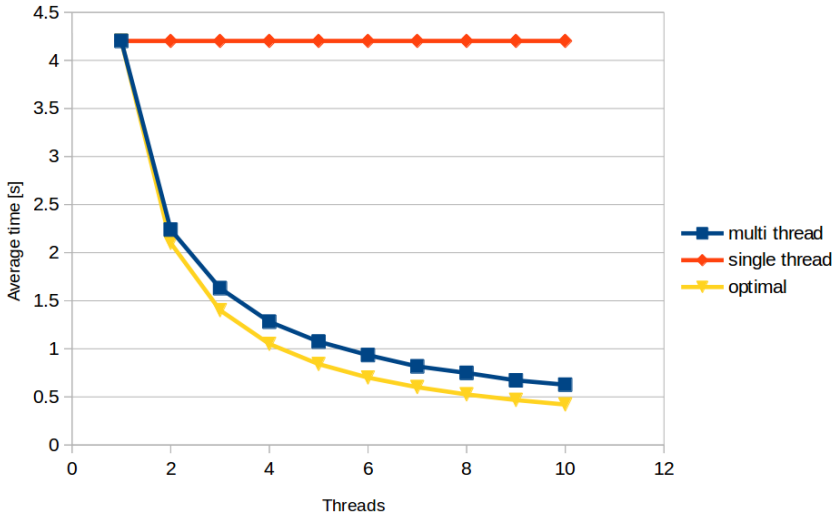
Paralelní programování

Měření zrychlení

Je dané zrychlení dostatečné? Můžeme být rychlejší?

- V optimálním případě se paralelní verze zrychluje proporčně s počtem jader

Vláken	1	2	3	4
Čas	x	x/2	x/3	x/4



Často vyjádřeno jako zrychlení:

$$S = \frac{T_{serial}}{T_{parallel}}$$

Paralelní programování

Měření zrychlení

Můžeme se vždy dostat k lineárnímu zrychlení?

- Paralelní verze algoritmů mají (téměř) vždy další režii
  - spouštění vláken
  - zámky
  - synchronizace
  - …
- Program/algoritmus často vyžaduje určitou sériovou část
  - Nechť jsme schopni přepsat 90% kódu s lineárním zrychlením
  - $S = \frac{T_{serial}}{0.9 \times \frac{T_{serial}}{p} + 0.1 \times T_{serial}} \leq \frac{T_{serial}}{0.1 \times T_{serial}}$
  - To znamená, že pokud sériový program trvá 20 sekund, nikdy nedosáhneme zrychlení větší než 10

Amdahlův zákon

Pthreads vs. C++ vs. OpenMP

Ochutnávka (pthreads)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

const int thread_count = 10;
void* Hello(void* rank);

int main(int argc, char* argv[]) {
   long thread;
   pthread_t *thread_handles;
   thread_handles = (pthread_t*)malloc(thread_count * sizeof(pthread_t));
   for (thread = 0; thread < thread_count; thread++)
      pthread_create(&thread_handles[thread], NULL,
                     Hello, (void *) thread);
   printf("Hello from the main thread\n");
   for (thread = 0; thread < thread_count; thread++)
      pthread_join(thread_handles[thread], NULL);

   free(thread_handles);
   return 0;
}

void* Hello(void* rank) {
   long my_rank = (long) rank;
   printf("Hello from thread %ld of %d\n", my_rank, thread_count);
   return NULL;
}
```

Pthreads vs. C++ vs. OpenMP

Ochutnávka (C++11)

```
#include <iostream>
#include <thread>
#include <vector>

const int thread_count = 10;
void Hello(long my_rank);

int main(int argc, char* argv[]) {
   std::vector<std::thread> threads;
   for (int thread=0; thread < thread_count; thread++) {
      threads.push_back(std::thread(Hello, thread));
   }

   std::cout << "Hello from the main thread\n";

   for (int thread=0; thread < thread_count; thread++) {
      threads[thread].join();
   }

   return 0;
}

void Hello(long my_rank) {
   std::cout << "Hello from thread " << my_rank << " of " << thread_count << std::endl;
}
```

Nicolai Josuttis: “it is almost impossible to use it easily and right”

-std=c++20 (

Pthreads vs. C++ vs. OpenMP

Ochutnávka (C++20)

```
#include <iostream>
#include <thread>
#include <vector>

const int thread_count = 10;
void Hello(long my_rank);

int main(int argc, char* argv[]) {
   std::vector<std::thread> threads;
   for (int thread=0; thread < thread_count; thread++) {
      threads.push_back(std::jthread(Hello, thread));
   }

   std::cout << "Hello from the main thread\n";

   return 0;
}

void Hello(long my_rank) {
   std::cout << "Hello from thread " << my_rank << " of " << thread_count << std::endl;
}
```

Specifikace byla schválena v listopadu 2020. GCC 11 s -std=c++20 kompiluje většinu testů ze specifikace, podpora je ale “experimentální” .

Pthreads vs. C++ vs. OpenMP

Ochutnávka (OpenMP)

```
#include <iostream>
#include <vector>
#include "omp.h"

const int thread_count = 10;

void Hello() {
    int my_rank = omp_get_thread_num();
    int threads = omp_get_num_threads();
    std::cout << "Hello from thread " << my_rank << " of " << threads << std::endl;
}

int main(int argc, char* argv[]) {
#pragma omp parallel num_threads(thread_count)
    Hello();
    return 0;
}
```

- nutno překládat s přepínačem –fopenmp
  - (např. **g++ -fopenmp openmp-hello.cpp -o openmp-hello**)

Přehled paralelní části

- Základní úvod
  - Vlákna, synchronizace, mutexy
  - Pthread (již by jste měli znát), C++11 thready
- OpenMP
  - nadstavba nad C kompilátorem pro zjednodušení implementace paralelních programů
- Techniky dekompozice
- Datové struktury umožňující přístup vícero vláken
- Základní paralelní řadící algoritmy a vektorové instrukce
- Základní paralelní maticové algoritmy

Odpovídající státnicové otázky

Paralelní část

Hardwarová podpora pro paralelní výpočty: (super)skalární architektury, pipelining, spekulativní vyhodnocování, vektorové instrukce, vlákna, procesy, GPGPU. Hierarchie cache pamětí.

Komplikace v paralelním programování: souběh (race condition), uváznutí (deadlock), iluze sdílení (false sharing).

Podpora paralelního programování v C a C++: pthreads, thread, jthread, atomic, mutex, lock_guard.

Podpora paralelního programování v OpenMP: sériově-paralelní model uspořádání vláken (fork-join), paralelizovatelná úloha (task region), různé implementace specifikace. Direktivy parallel, for, section, task, barrier, critical, atomic.

Techniky dekompozice programu: statické a paralelní rozdělení práce. Threadpool a fronta úkolů. Balancování a závislosti (dependencies).

Techniky dekompozice programu na příkladech z řazení: quick sort, merge sort.

Techniky dekompozice programu na příkladech z numerické lineární algebry a strojového učení: násobení matice vektorem, násobení dvou matic, řešení systému lineárních rovnic.

Odpovídající státnicové otázky

Distribuovaná část

Úvod do distribuovaných systémů (DS). Charakteristiky DS. Čas a typy selhání v DS.

Detekce selhání v DS. Detektory selhání a jejich vlastnosti.

Čas a kauzalita v DS. Uspořádání událostí v DS. Fyzické hodiny a jejich synchronizace. Logické hodiny a jejich synchronizace.

Globální stav v DS a jeho výpočet. Řez distribuovaného výpočtu. Algoritmus pro distribuovaný globální snapshot. Stabilní vlastnosti DS.

Vzájemné vyloučení procesů v DS. Algoritmy pro vyloučení procesů a jejich vlastnosti.

Volba lídra v DS. Algoritmy pro volbu lídra a jejich vlastnosti.

Konsensus v DS. FLP teorém. Algoritmy pro distribuovaný konsensus.