Ústav radioelektroniky
Vysoké učení technické v Brně

Zvyšování početního výkonu procesorů

Mikroprocesorová technika a embedded systémy

Přednáška 10

doc. Ing. Tomáš Frýza, Ph.D.
Ing. Jan Prokopec, Ph.D.

listopad 2012

Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

Vyrovnávací paměť Cache

Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

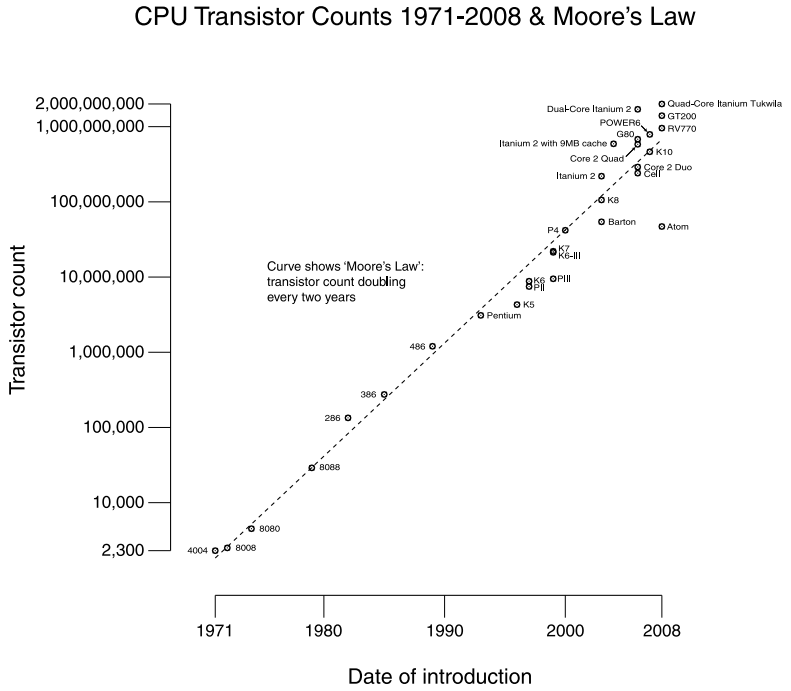
High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

Vyrovnávací paměť Cache

Moorův zákon

"Cramming more components onto integrated circuits", Electronics Magazine 19 April 1965: The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer



11/09 AMD Opteron 2435: 900 mil. transistors, 6 cores, 2 600 MHz, 45 nm

11/09 Intel Core i7 975 XE: 731 mil. transistors, 4 cores, 3 333 MHz, 45 nm

11/10 Nvidia GTX 480: 3 000 mil. transistors, 480 cores, 700 MHz, 40 nm

11/11 Intel Core i7 3960X, 2 200 mil. transistors, 6/12 cores, 3 300(3 900) MHz, 32 nm (22 nm)

Vývoj mikroprocesorové techniky

- Vývoj v mikroprocesorové technice se ubírá dvěma směry, první dosahuje extrémních výkonů (PC), kdy využitelné jsou pouze poslední produkty a druhý směr usiluje o nízkou spotřebu, cenu, omezený výkon (MCU).

Microprocessor	Year of introduction	No. of Transistors
4004	1971	2 300
8008	1972	2 500
8080	1974	4 500
8086	1978	29 000
Intel286	1982	134 000
Intel386	1985	275 000
Intel486	1989	1 200 000
Intel Pentium	1993	3 100 000
Intel Pentium II	1997	7 500 000
Intel Pentium III	1999	9 500 000
Intel Pentium 4	2000	42 000 000
Intel Itanium	2001	25 000 000
Intel Itanium 2	2003	220 000 000
Intel Itanium 2 (9 MB cache)	2004	592 000 000

Způsoby zvyšování výkonu mikroprocesorů

- Výkon mikroprocesorů lze ovlivnit několika prostředky, např.: změnou výrobní technologie, efektivním funkčním uspořádáním, přídavnými jednotkami, či zřetězením instrukcí.
- Dopady rozdílných výrobních technologií:
  - Vyšší hustota integrace umožňuje konstrukci většího počtu prvků na jednotku plochy (2000 - 180 nm, 2009 45 nm, 2010 32 nm, 2012 22 nm):
    - Menší parazitní kapacity mezi vodiči/prvky; což způsobuje menší přepínací ztráty a současně kratší dobu přechodného děje.
    - Může růst frekvence hodinového signálu; ale ztrátový/přepínací výkon celé součástky závisí na f_{CPU}, tj. při vyšší frekvenci je také procesor více tepelně zatížen.
    - Větší čistota křemíkového substrátu umožňuje konstrukci větších čipů.
    - Více elementárních prvků na čipu způsobuje také větší ztrátový výkon.
  - Ztrátový výkon lze ovlivnit snížením napájecího napětí a f_{CPU}.

Způsoby zvyšování výkonu mikroprocesorů

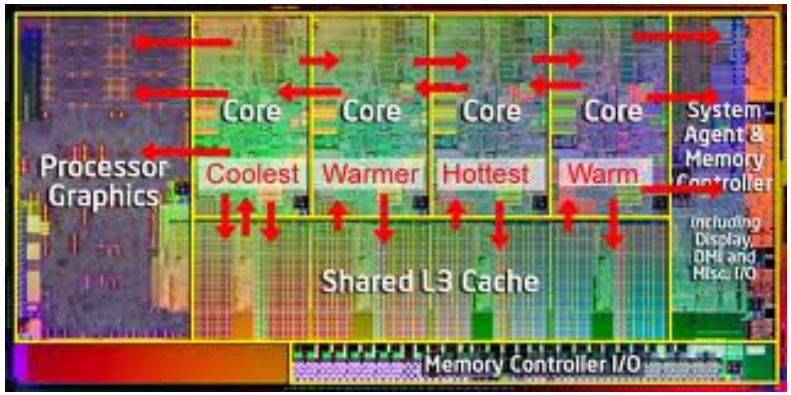
- Kolem roku 2000 začíná éra přetaktování procesorů
- u starších typů (AMD thunderbird) HW zásah do procesoru
- výrazné vylepšení poměru cena-výkon
- pro přetaktování není vhodná každa řada procesoru
- Pentium 4 - ztrátový výkon TDP 140 W, C2D 65W - přetaktování na 5 GHz
- nároky na chlazení - měď, heatpipe, vodník, dusík
- pouze pro PC
- v oblasti serverů nemožné (spolehlivost, rack mount, ...)

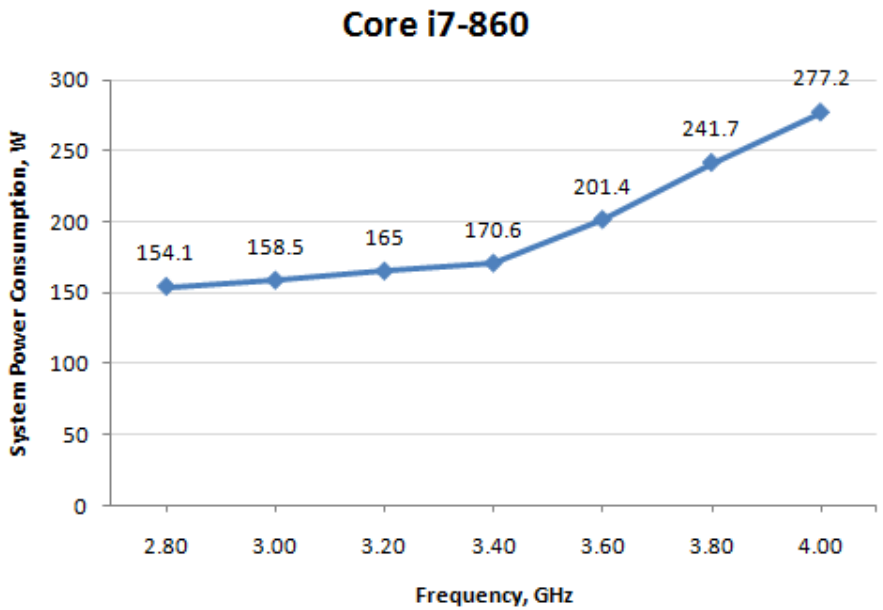
Thermal Design Power

- Thermal Design Power

$$P \propto CU^2 f$$

- limitation of max temperature of Silicon die
- as example, Intel i7 860 processor, die 296 mm^2, system power consumption
- not possible solution for server segment

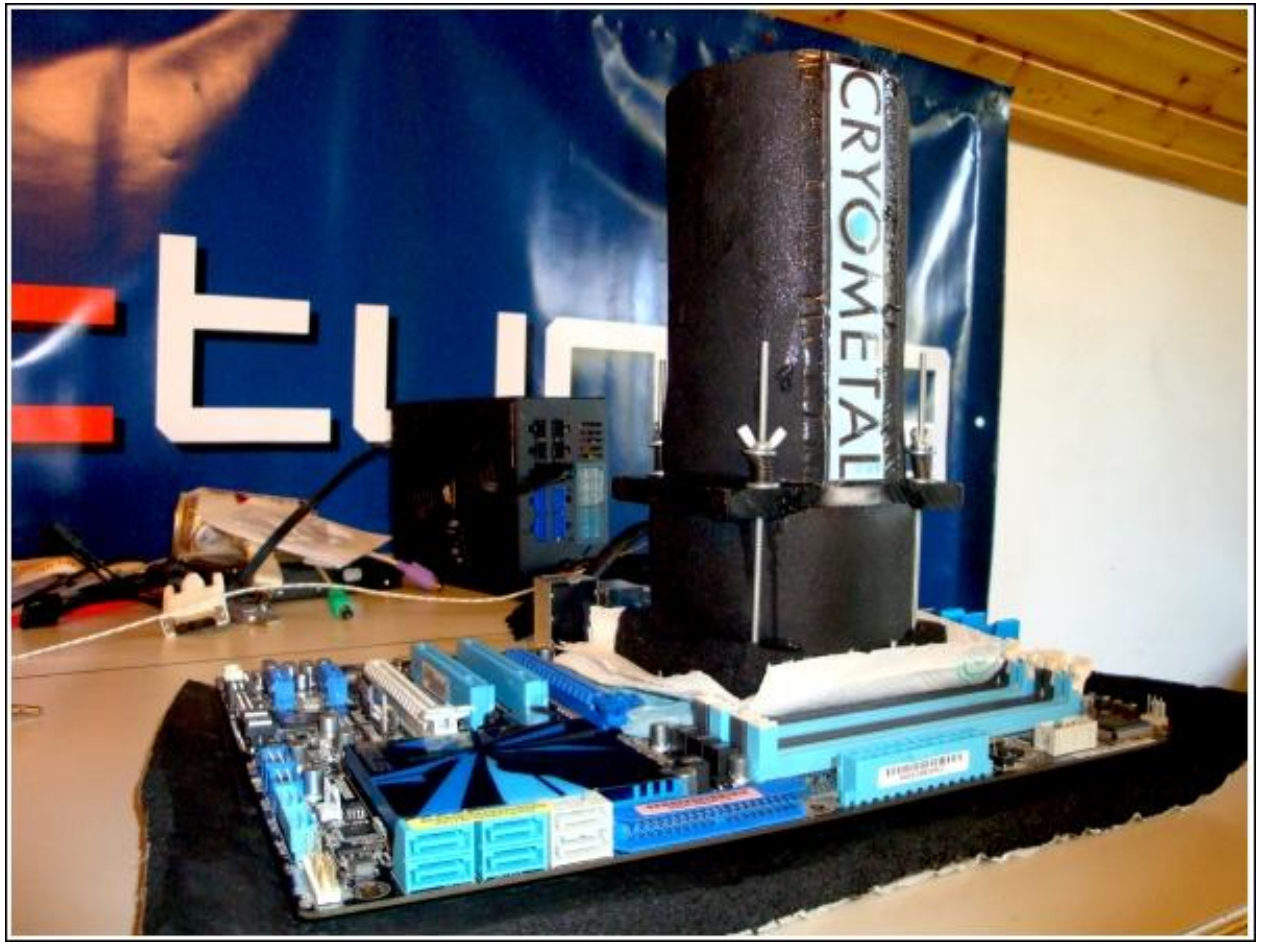




Způsoby zvyšování výkonu mikroprocesorů

Obrázek: Vodní chlazení

Způsoby zvyšování výkonu mikroprocesorů



Obrázek: Dusík

Způsoby zvyšování výkonu mikroprocesorů

- Zvýšení početního výkonu změnou funkční struktury:
  - Možnost umístit mikroprocesor + paměti + periférie na jediný čip.
  - Rozšíření datové (8 → 16, 32, 64 bitů) a adresní sběrnice (16 → 20, . . ., 32 bitů) efektivně využívá velikost křemíkového substrátu:
    - Lze tak zpracovávat větší operandy.
    - Šířka bitového slova koresponduje maximální hodnotu, kterou dokáže mikroprocesor zpracovat během jedné operace (255@8bitové, 65 536@16bitové).
    - Hodnoty větší je nutné nejprve rozdělit a následně počítat v několika krocích (viz 2bytové operace na AVR).

  Pozn.: Procesory Intel 8086 (16bitová DB) a 8088 (8bitová DB) mají shodnou instrukční sadu. Přesto, při shodné frekvenci f_{CPU} může 8086 pracovat až 2× rychleji.
- Přechod na paralelní činnost:
  - Jádro procesoru obsahuje větší počet funkčních jednotek (včetně sběrnic, viz architektura VLIW).
  - Možnost výkonu většího počtu instrukcí současně.
  - Řídicí jednotka umožňuje načtení většího počtu instrukcí a následně je přeskládá pro efektivnější výkon.
- Koprocesory:
  - Dříve byly procesory doplňovány externími matematickými koprocesory, které vykonávaly numerické operace (ještě u 80386).
  - Později se koprocesory integrovaly přímo na čip procesoru, což opět způsobuje větší početní výkon.

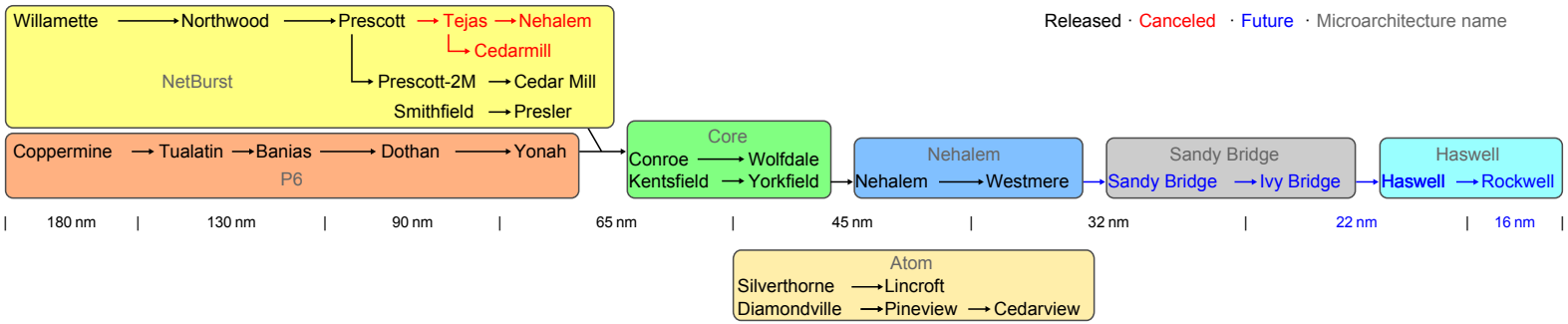
Způsoby zvyšování výkonu mikroprocesorů

- Přechod na paralelní činnost (2005):
  - Intel zaostává s technologií Netburst
  - Netburst - technologie založená na hlubokém větvení vykonávání instrukcí
  - předpoklad až 5 GHz, ale obrovské TDP 140 W na 3 GHz
  - objevuje se HyperThreading - "softwarové" dvoujádro
  - Hyper threading musí podporovat OS
  - AMD Athlon X2 - nativní dvoujádro
  - efektivní pro náročné výpočty (paralelní zpracování)
  - ALE musí být optimalizované aplikace - problém!
  - nesmí být závislost vláken mezi sebou, jinak zpomalení
- Jen víc a houšť jader ...
  - Intel Core, Core 2 Duo - podobná Pentium Pro z roku 1995
  - dvoujádra, v nejvyšších řadách včetně Hyperthreadingu - virtuální 4 jádro
  - postupně návrat na technologickou špičku
  - Core 2 Quad - 2 x 2 Core 2 Duo na jednom čipu

  a nebo ...
  - AMD Athlon X4, problémy s výrobou, šikovný tah s X3
  - X3 "vadná" 4 jádra z výroby, která ale šla někdy odemknout na 4 jádra
  - pořád schází podpora software
  - desktop segment má problém s malou RAM (řadič pamětí - 4 sloty)

Způsoby zvyšování výkonu mikroprocesorů

- současnost u Intelu ...
  - Intel Core i7 - výkon super PC z roku konce 90 let v desktopu
  - s hyperthreadingem 8 jader
  - chipset s podporou 16 GB RAM
  - ale není podpora software, nicméně, více instancí programu Matlab (např.)
  - výhodné pro virtualizaci



Obrázek: Technology roadmap Intel

- AMD nespí ...
  - AMD zejména serverová oblast - viz výpočetní server
  - 2009 - Opteron 4 jádra, vysoká propustnost pamětí
  - 2010 - Opteron 6 jader, následují nejvyšší řady 6000 - 12 nebo 8 jader
  - nástup masivní virtualizace v serverové oblasti

Odskok - výpočetní server UREL

- konfigurace je zvolena pro výpočty z oblasti numerických metod pro více software
- není výhodné vytvořit cluster se sdílenými zdroji
- konfigurace např. 2 x Quad Core Opteron 2.7 GHz, 32 GB RAM, 72 GB SAS disk + 1 TB diskové pole
- nebo 2 x Quad Core Opteron 2.1 GHz, 16 GB RAM, 72 GB SAS disk + 1 TB diskové pole - to je sdílené
- lze pro řešení složitějších úloh přiřazovat HW zdroje konkrétním výpočtům
- experimenty s clustrem pomocí virtualizovaného HW (96 GB RAM, 6CPU, 28 jader)

Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

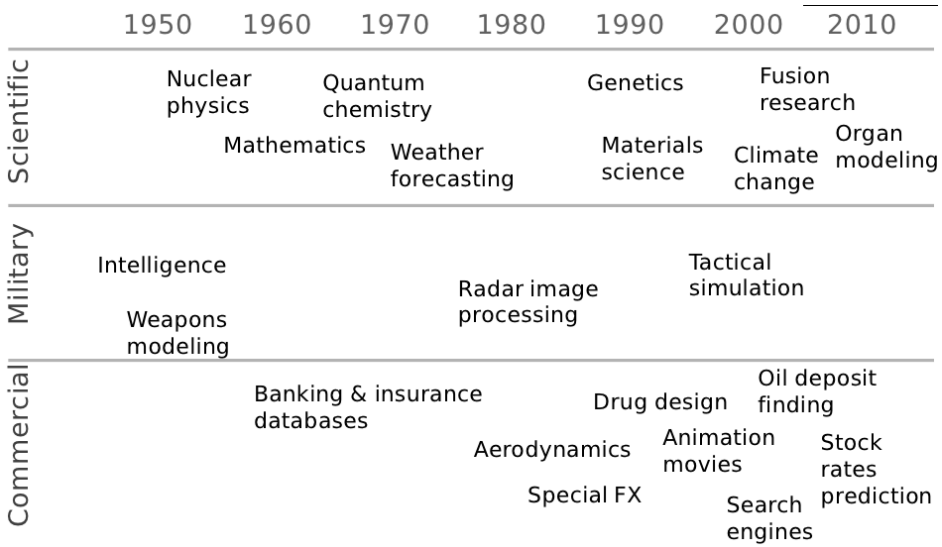
Vyrovnávací paměť Cache

High performance computing

Definice

High performance computing (HPC) stands for scientific parallel computing on a high-performance system, i.e. systems listed on the Top500 list (`http://top500.org/`).

- Multidisciplinary problems
- Coupled applications
  - Full simulation of engineering systems
  - Full simulation of biological systems
  - Astrophysics
  - Materials science
  - Bio-informatics, proteomics, pharmaco-genetics
  - Scientifically accurate 3D functional models of the human body
  - Biodiversity and biocomplexity
  - Climate and atmospheric research
  - Energy
  - Digital libraries for science and engineering
- Large amount of data
- Complex mathematical models



Obrázek: HPC through the ages.

Current HPC trend

- Top500 project ranks and details the 500 (non-distributed) most powerful known computer systems in the world. (Ranking of supercomputers according to the LINPACK benchmark; R_{max} – Maximal LINPACK performance achieved.)
- Top500: some facts

1976 Cray 1 installed at Los Alamos National Laboratory: peak performance 160 MegaFlop/s (10^6 flop/s)

1993 (1° Edition of Top500 list was published at the Supercomputing Conference in Mannheim, June 1993.) N.1: CM-5/1024 system with $R_{max} = 59.7$ GFlop/s (10^9 flop/s)

1997 Teraflop/s barrier (10^{12} flop/s)

2008 Petaflop/s barrier (10^{15} flop/s): Roadrunner (Los Alamos National Lab) $R_{max} = 1\,026$ Gflop/s. Hybrid system: 6 562 processors dual-core AMD Opteron accelerated with 12 240 IBM Cell processors

2011 $R_{max} = 10.5$ Pflop/s. K computer (SPARC64 VIIIfx 2.0GHz) RIKEN Japan

- 62% of the systems on the top500 use processors with six or more cores
- 39 systems use GPUs as accelerators (35 NVIDIA, 2 Cell, 2 ATI Radeon)
- commodity Linux clusters occupying a great share of the Top500 list

???? Ostrava, Czech Republic

Center for Scientific Computing, Finland

Figure: CSC (Center for Scientific Computing), Finland; Cray XT5/XT4 QC 2.3 GHz; Top500 Nov-2011 no. 202. Total cores: 10 864. Maximal performance $R_{max} = 76{,}5$ Tflop/s. Processors: Opteron Quad Core 4C 2.3 GHz. Processor technology: AMD x86_64.

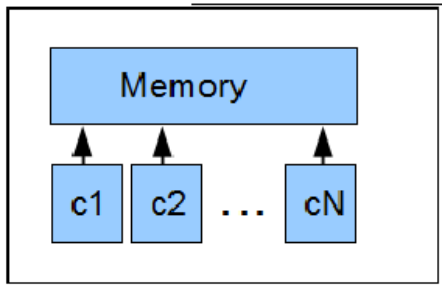
HPC systems evolution

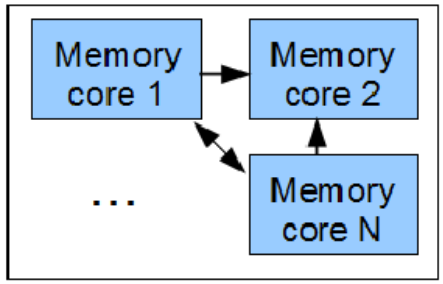
- Vector Processors
  - Cray-1
- SIMD, Array Processors
  - Goodyear MPP, MasPar 1 & 2, TMC CM-2
- Parallel Vector Processors (PVP)
  - Cray XMP, YMP, C90 NEC Earth Simulator, SX-6
- Massively Parallel Processors (MPP)
  - Cray T3D, T3E, TMC CM-5, Blue Gene/L
- Commodity Clusters
  - Beowulf-class PC/Linux clusters
  - Constellations
- Distributed Shared Memory (DSM)
  - SGI Origin
  - HP Superdome
- Hybrid HPC Systems
  - Roadrunner
  - Chinese Tianhe-1A system
  - GPGPU systems (General-purpose computing on graphics processing units; `http://gpgpu.org/`)

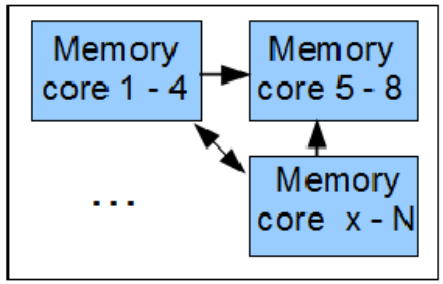
Obrázek: Ancient Beowulf cluster.

Parallel computers; Parallel programming models

- Shared memory: all the cores can access the whole memory.
- Distributed memory: all the cores have their own memory and communication is needed in order to access the memory of other cores.
- Current supercomputers combine the distributed memory and shared memory approaches.







- Message passing
  - Can be used both in distributed and shared memory computers.
  - Programming model allows for good parallel scalability.
  - Programming is quite explicit.
- Threads (pthreads, OpenMP)
  - Can be used only in shared memory computers.
  - Limited parallel scalability.
  - "Simpler" /less explicit programming.
- Hybrid programming
  - Threads inside a node, message passing between nodes
  - Can enable scaling to extreme core counts (>10 000)

Parallel computing concepts

- Parallel computation = executing tasks concurrently
  - A task encapsulates a sequential program and local data, and its interface to its environment
  - Data those of other tasks is remote
- Data dependency = computation in one task requires data in another task in order to proceed

- Synchronization
  - coordination of processes for maintaining correct runtime order and for keeping data coherent
- Load balance
  - distribution of workload to different cores
- Parallel overhead
  - additional operations which are not present in serial calculation
  - synchronization, redundant computations, communications

Message passing interface (MPI)

- MPI is an application programming interface (API) for communication between separate processes
  - The most widely used approach for distributed parallel computing
- MPI programs are portable and scalable
- MPI is flexible and comprehensive
  - large (over 120 procedures)
  - concise (often only 6 procedures are needed)
- MPI standardization by MPI Forum
  - ...
  - MPI 2.2 Released September 4, 2009
  - MPI 3.0 Released September 21, 2012
  - `http://www.mpi-forum.org/`
- Execution model
  - Parallel program is launched as set of independent, identical processes
  - The same program code and instructions
  - Can reside in different nodes ... or even in different computers
  - The way to launch parallel program is implementation dependent
    `mpirun`, `mpiexec`, `aprun`, `poe`, ...

- MPI runtime assigns each process a rank
  - identification of the processes
  - ranks start from 0 and extent to N-1
- Processes can perform different tasks and handle different data basing on their rank

```
1  ...
2  if( rank == 0 ){
3      ...      // master code
4  }
5  if( rank != 0 ){
6      ...      // slave(s) code
7  }
```

Data model; Routines of the MPI library; Programming MPI

- All variables and data structures are local to the process
- Processes can exchange data by sending and receiving messages

- Obtaining information about the communicator
  - number of processes,
  - rank of the process
- Communication between processes
  - sending and receiving messages between two processes
  - sending and receiving messages between several processes
- Synchronization between processes
- Advanced features

- MPI standard defines interfaces to C and Fortran programming languages
- C call convention

```
1  rc = MPI_Xxxx( parameter, ... ) ;
2  // some arguments have to passed as pointers
```

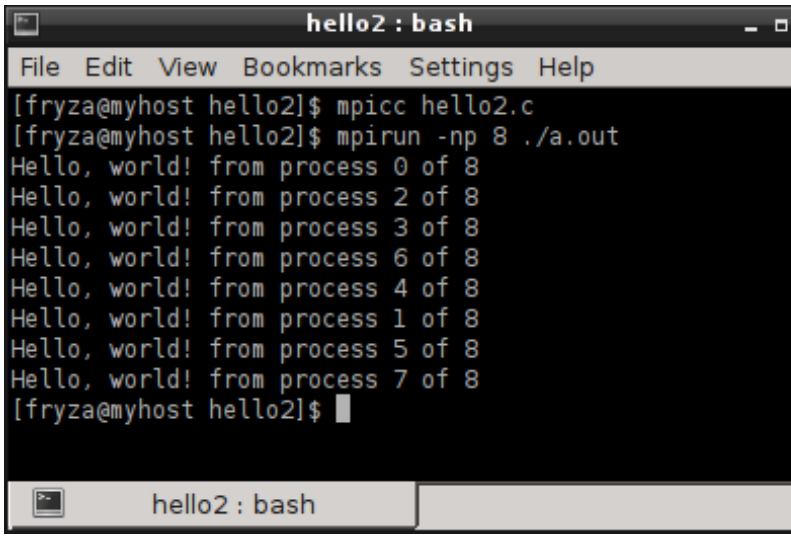
- Fortran call convention

```
1  CALL MPI_XXXX( parameter, ..., rc )
2  ! return code in the last argument
```

- How to exploit parallel possibilities on desktop? Just install `openmpi` package (`http://www.open-mpi.org/`)!

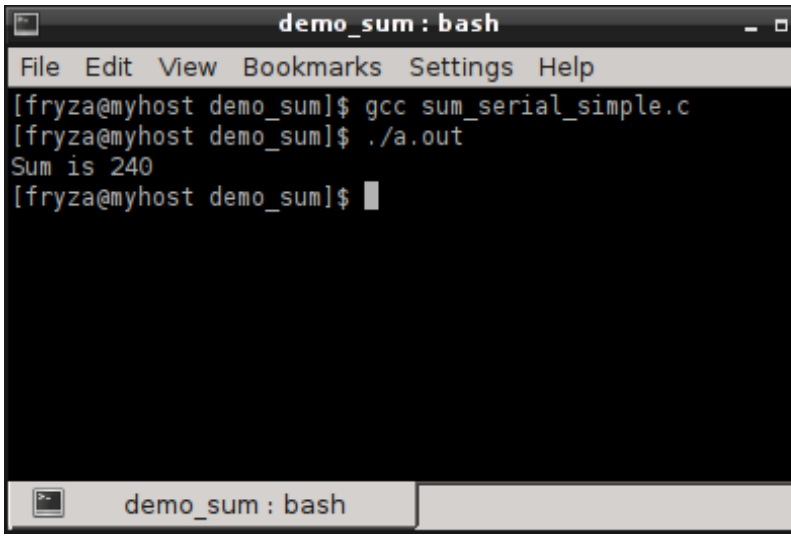
Hi there from many ...

```c
#include <stdio.h>
#include <mpi.h>

int main( int argc, char *argv[] )
{
    int rank, size ;

    MPI_Init( &argc, &argv ) ;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );

    printf( "Hello, world! from process %d of %d\n", rank,size ) ;

    MPI_Finalize() ;

    return( 0 ) ;
}
```



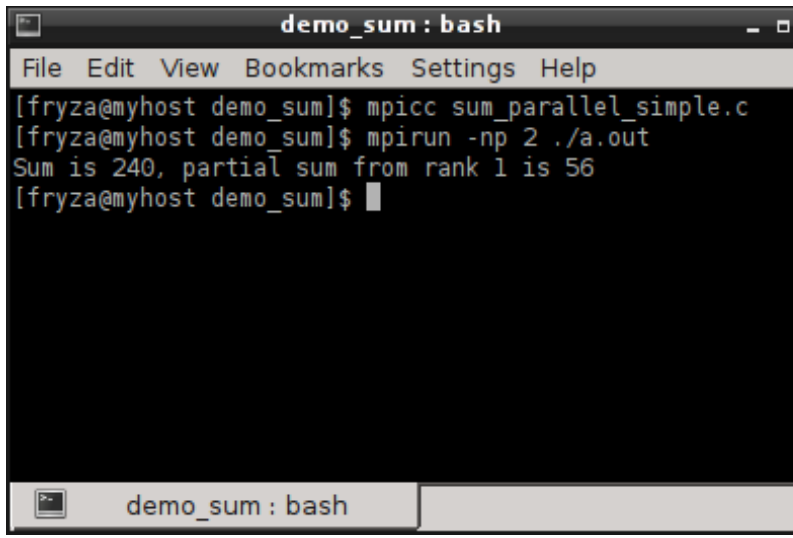
Sum of array – sequential version

```c
#define N   16
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    int i ;
    double *array ;
    double sum ;

    // allocate array
    array = malloc( sizeof(double) * N ) ;

    // initialize array
    for( i=0; i<N; i++ ){
        array[i] = 2.0*i ;
    }

    // sum array
    sum = 0 ;
    for( i=0; i<N; i++ ){
        sum += array[i] ;
    }

    // print result
    printf( "Sum is %g\n", sum ) ;

    free( array ) ;
    return( 0 ) ;
}
```

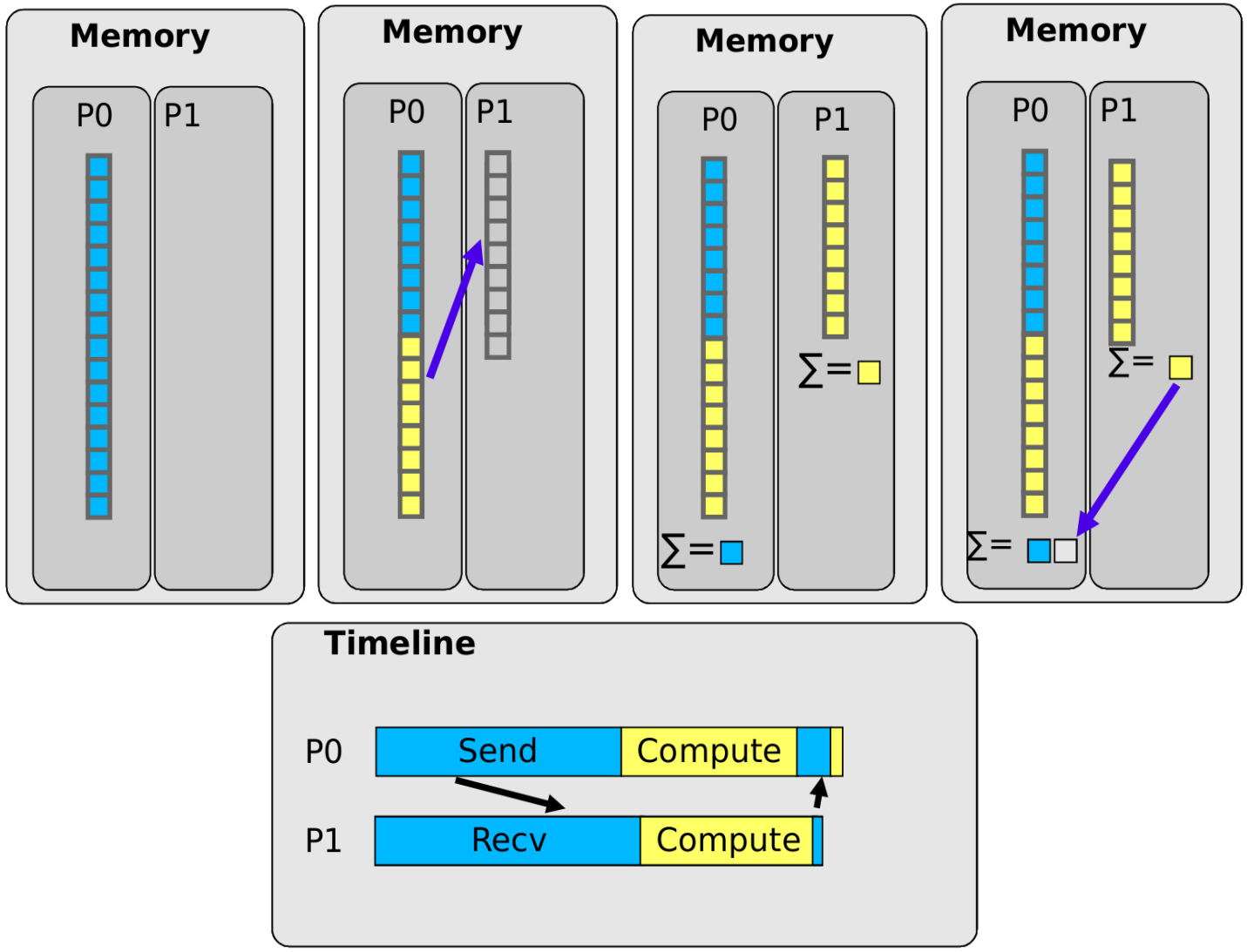


Sum of array – parallel version

```
1   ...
2   #include <mpi.h>
3
4   int main( int argc, char *argv[] )
5   {
6       ...
7       MPI_Init( &argc, &argv ) ;
8       MPI_Comm_rank( MPI_COMM_WORLD, &rank );
9
10      // send & receive upper half of array
11      if( rank == 0 ) MPI_Send( ... ) ;
12      if( rank == 1 ) MPI_Recv( ... ) ;
13
14      // compute local sums
15      psum = 0 ;
16      for( i=0; i<N/2; i++ ){
17        psum += array[i] ;
18      }
19
20      // send local sum from 1 to 0
21      if( rank==0 ) MPI_Recv( &sum, ... ) ;
22      if( rank==1 ) MPI_Send( &psum, ... ) ;
23
24      // compute results on 0 and print it
25      if( rank==0 ){
26        sum += psum ;
27        printf( "Sum is %g, partial sum from ↩
                rank 1 is %g\n", sum,psum ) ;
28      }
29
30      MPI_Finalize() ;
31      return( 0 ) ;
32  }
```

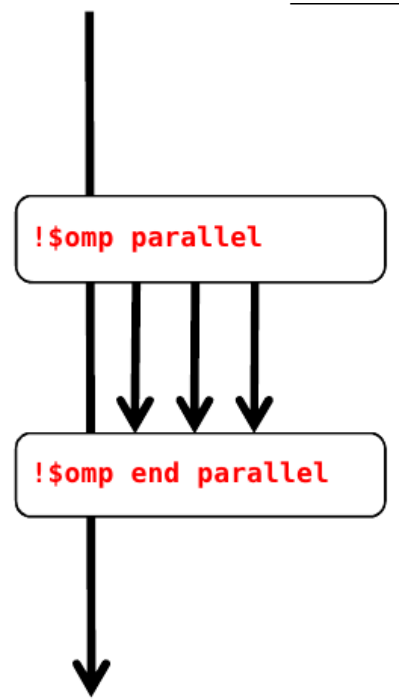


Parallel sum – message passing



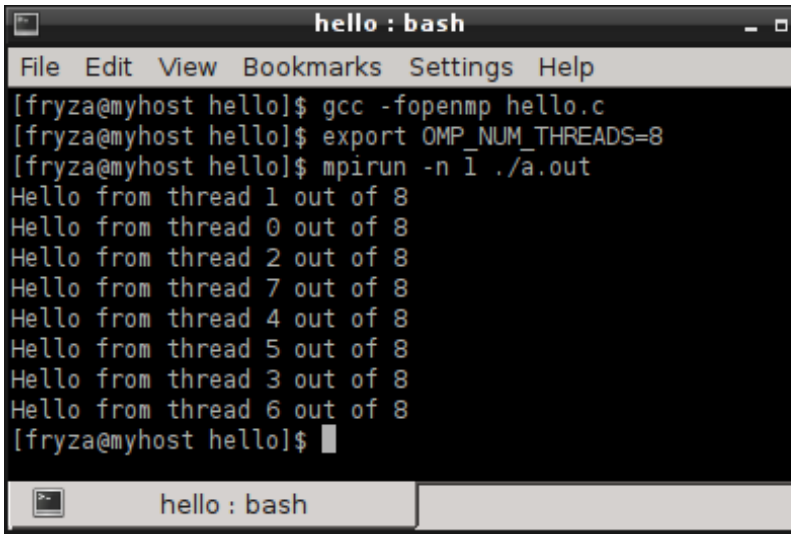
Why would you want to learn OpenMP (Open Multiprocessing)?

- An OpenMP API that can be used for multi-threaded shared memory parallelization
- Fortran 77/9X and C/C++ are supported
- Latest Official OpenMP Specifications:
  - Version 3.1 Complete Specifications (July 2011)
- OpenMP parallelized program can be run on your many-core workstation
- Enables one to parallelize one part of the program at a time
- Serial and OpenMP versions can easily coexist
- Hybrid programming
- Three components of OpenMP
  - Compiler directives: expresses shared memory parallelization; preceded by sentinel, can compile serial version
  - Runtime library routines: small number of library functions; can be discarded in serial version via conditional compiling
  - Environment variables: specify the number of threads, etc.
- Starts a parallel region
  - Prior to it only one thread, master
  - Creates a team of threads: master+slave threads
  - At end of block is a barrier and all shared data is synchronized

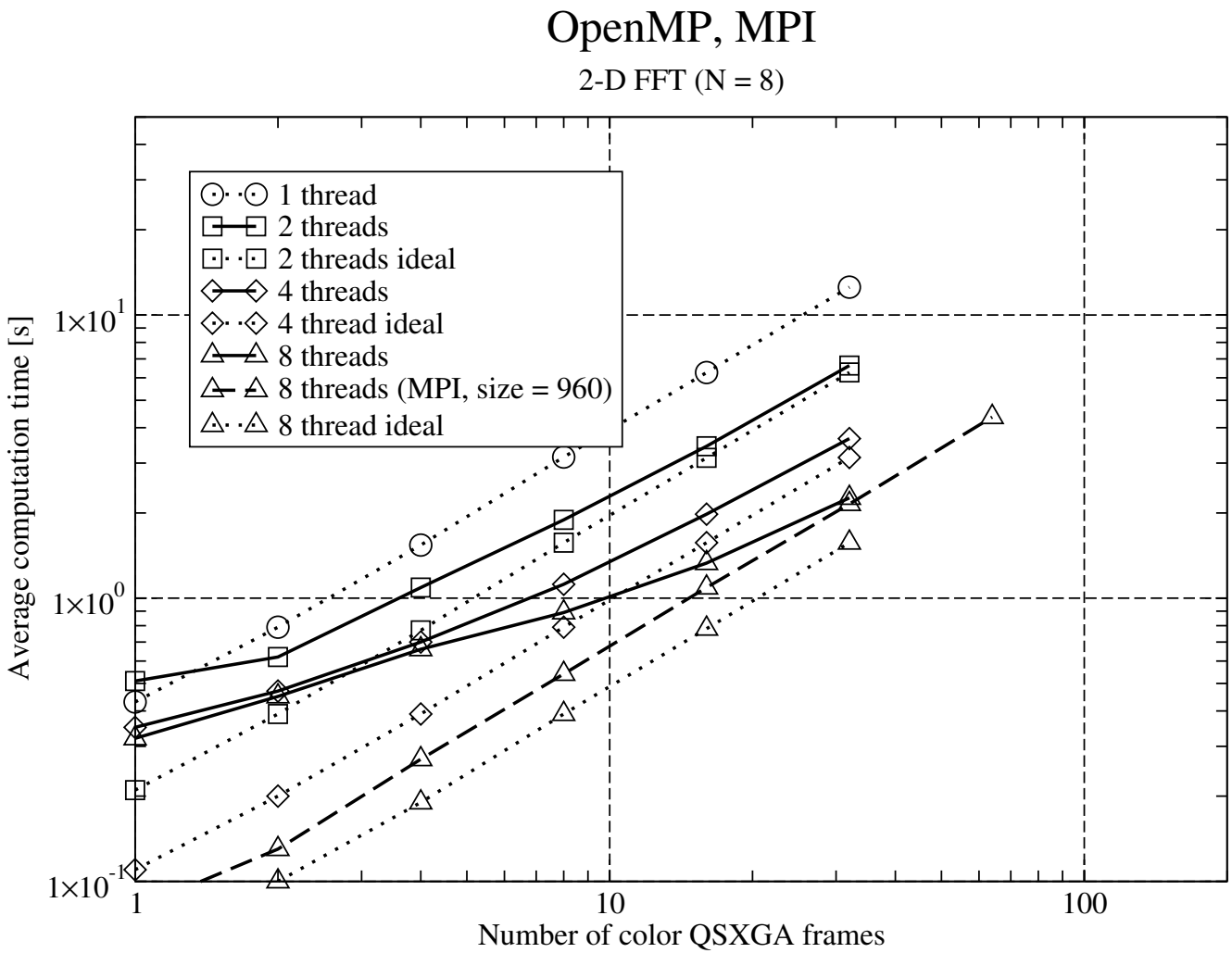


Hello world in parallel

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main( int argc, char *argv[] ){
5      int omp_rank ;
6      int omp_threads ;
7
8  #pragma omp parallel private( omp_rank,↩
       omp_threads )
9  {
10     omp_rank = omp_get_thread_num() ;
11     omp_threads = omp_get_num_threads() ;
12
13     printf( "Hello from thread %d out of %d↩
           \n", omp_rank,omp_threads ) ;
14 }
15     return( 0 ) ;
16 }
```



Computing performance MPI vs. OpenMP

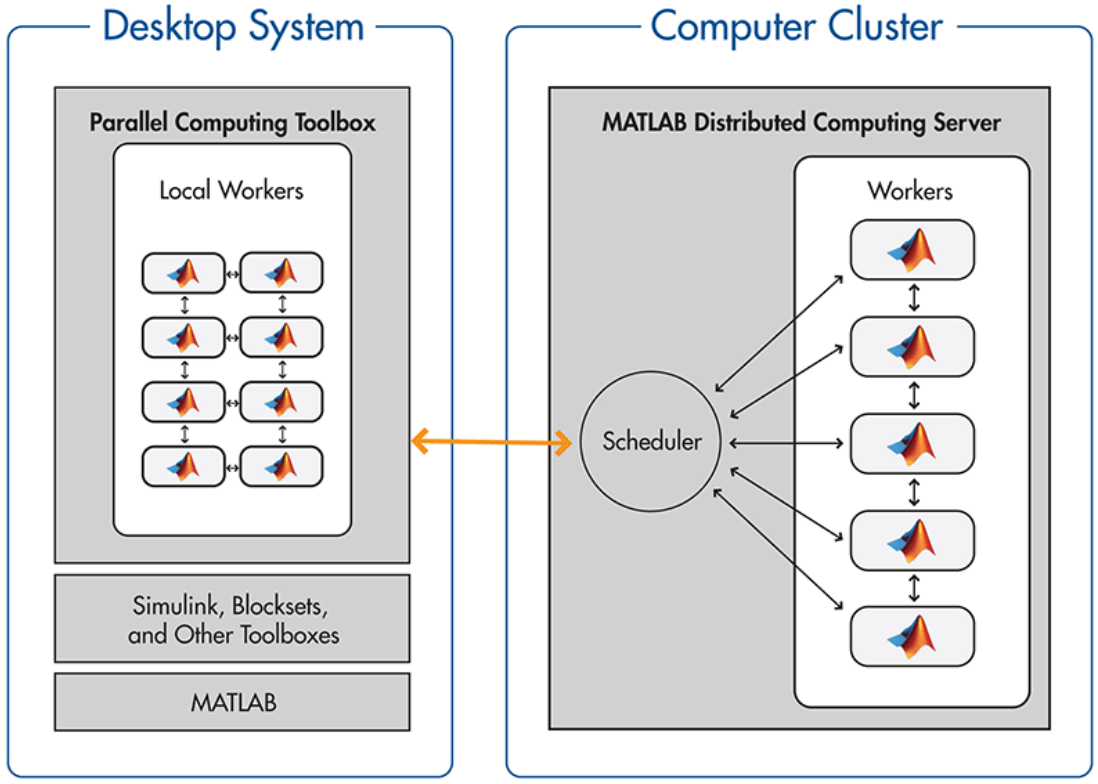


Obrázek: Average computation time for two-dimensional FFT OpenMP implementation with varying transformed frames and threads number ($N = 8$, $f_{CPU} = 2.7$ GHz, QSXGA color frames: 2,560×2,048 pixels).

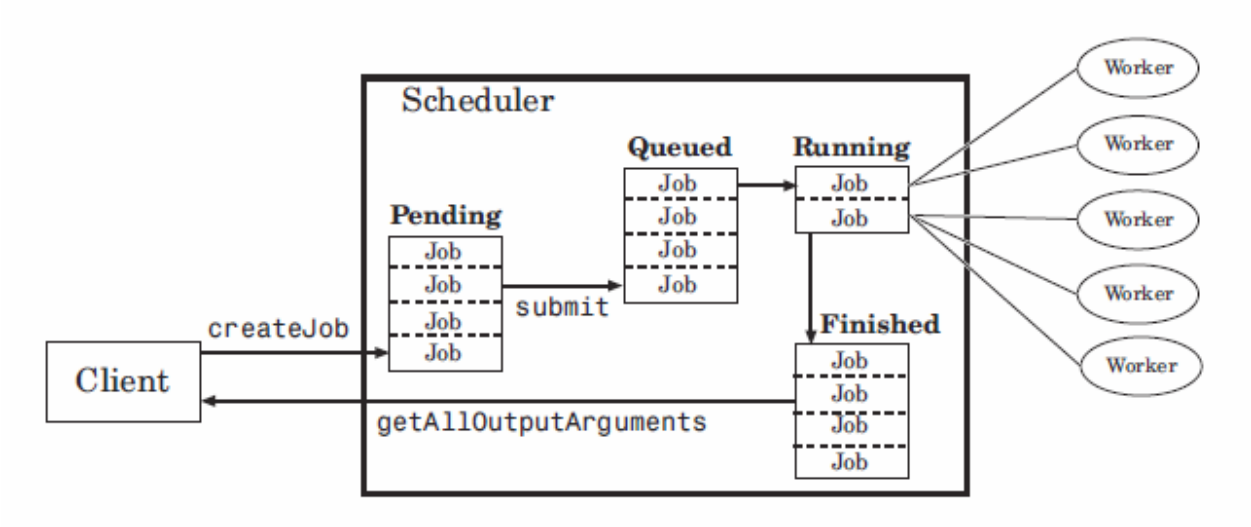
Matlab parallel and distributed processing architecture

Terms

- Client
- Worker
- JobManager
- Cluster
- MDCE
- Paralell toolbox
- Distributed computing toolbox



Job processing



- Client communicate with JobManager
- Scheduler (`mpiexec,torque,...`)
- Scheduler pass the `task` on `worker`
- and returns results to client

Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

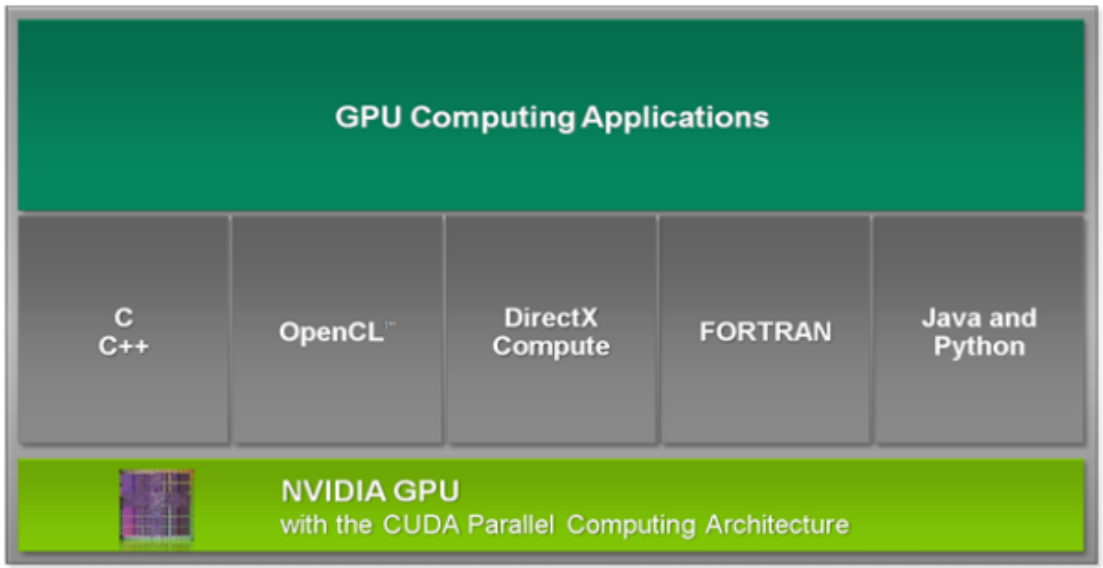
High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

Vyrovnávací paměť Cache

CUDA

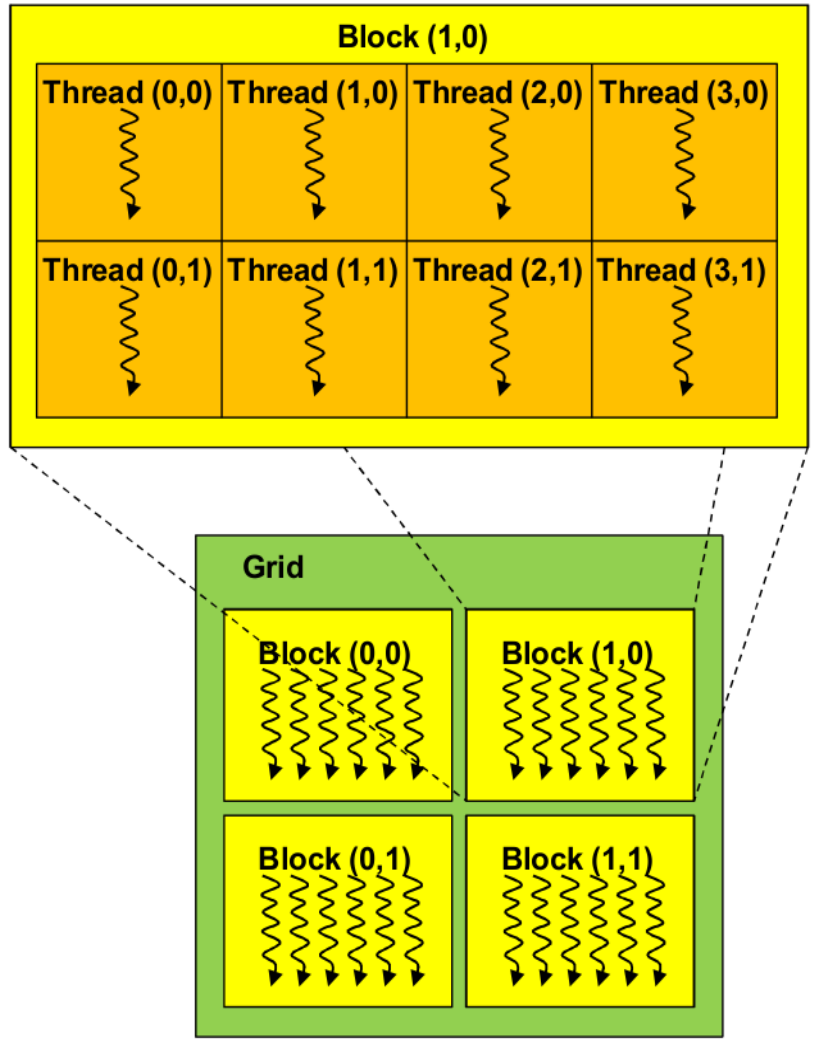
- Compute Unified Device Architecture (CUDA)
- CUDA C is a C/C++ language extension for GPU programming
  - PGI has developed similar Fortran 2003 extension
- CUDA API is the most up-to-date GPGPU programming interface for NVIDIA GPUs
- CUDA programs can be run on NVIDIA GPUs from different hardware generations (`http://www.nvidia.com/object/cuda_home_new.html`)
- Since MATLAB 2010 CUDA is supported



Obrázek: Graphics processing unit computing applications.

CUDA Programming Model; CPU and GPU Memories

- GPU accelerator is called **device**, CPU is **host**
- GPU code (kernel) is launched and executed on the device by several threads
- Threads are grouped into thread blocks
  - Dimensions are set at kernel launch
- Program code is written from a single thread's point of view
  - Each thread can diverge and execute a unique code path (can cause performance issues)

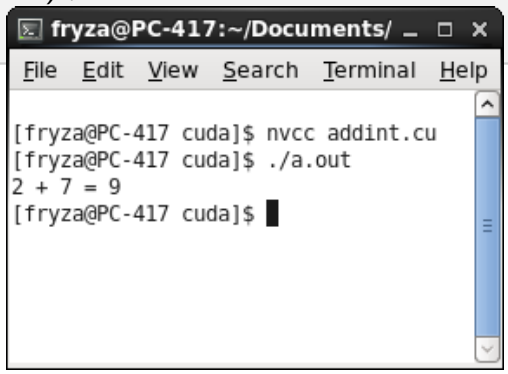


- Host and device have separate memories; host manages the GPU memory
- Usually one has to
  - Copy (explicitly) data from host to the device
  - Execute the GPU kernel
  - Copy (explicitly) the results back to the host
- Data copies between host and device use the PCI bus with very limited bandwidth $\rightarrow$ minimize the transfers!

Device Code

- C functions with restrictions
  - Can only dereference pointers to device memory
  - No static variables, no recursion
  - No variable number of arguments
- Functions must be declared with a qualifier
  - __global__ Kernel, called from CPU
    Cannot be called from GPU
    Must return void
  - __device__ Called from __device__ and __global__ funcs
    Can not be called from CPU
  - __host__ Can only be called by CPU
    Can be combined with __device__ qualifier

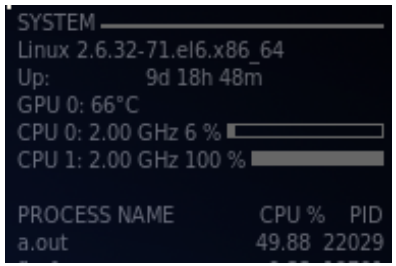
```
1  #include <stdio.h>
2
3  __global__ void add( int a, int b, int *c )
4  {
5      *c = a + b ;
6  }
7
8  int main( void )
9  {
10     int c ;
11     int *dev_c ;
12     cudaMalloc( (void**)&dev_c, sizeof(int)) ;
13
14     // run kernel in 1 block and 1 thread
15     add<<<1,1>>>( 2, 7, dev_c ) ;
16
17     // copy result from device to host
18     cudaMemcpy( &c, dev_c, sizeof(int), ↩
           cudaMemcpyDeviceToHost ) ;
19
20     printf( "2 + 7 = %d\n", c ) ;
21     cudaFree( dev_c ) ;
22
23     return( 0 );
24 }
```



Jacobi iteration by CPU and GPU

- This example solves the Poisson's equation in 2D using Jacobi iteration
  - CPU: $f_{CPU} = 2\,\text{GHz}$; 1 core
  - GPU: NVIDIA Quadro 4000; $f_{GPU} = 950\,\text{MHz}$; Maximal threads per block: 1024

```
File  Edit  View  Search  Terminal  Help
[fryza@PC-417 cuda]$ nvcc -arch=sm_20 ex3.cu
[fryza@PC-417 cuda]$ ./a.out
100 0.00972858
200 0.00479832
300 0.00316256
400 0.00234765
500 0.00186023
600 0.00153621
700 0.0013054
800 0.00113277
900 0.000998881
1000 0.000892078
CPU Jacobi: 20.8203 seconds, 1000 iterations
100
200
300
400
500
600
700
800
900
1000
GPU Jacobi: 0.151043 seconds, 1000 iterations
[fryza@PC-417 cuda]$
```



Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

Vyrovnávací paměť Cache

Zřetězené zpracování instrukcí, pipelining

- Podstatné zvýšení početního výkonu je docíleno zřetězeným zpracování instrukcí, tzv. pipelining:
  - Podstatou pipeliningu je skutečnosti, že zpracování každé instrukce lze rozdělit do několika navazujících fází, např.:
    - **F** (Fetch) – načtení instrukce z programové paměti nebo z vyrovnávací paměti cache.
    - **D1** (Decode1) – dekódování instrukce, určí se její typ.
    - **D2** (Decode2) – výpočet adresy operandů.
    - **E** (Execution) – vlastní provedení instrukce.
    - **W** (Write) – zápis výsledků operace.
  - Každou z fází lze provést pomocí samostatné podjednotky. Po skončení úkolu předá podjednotka svůj výsledek následující podjednotce a sama zpracovává část následující instrukce.
  - Vykazuje větší využití sběrnic.
  - Jsou kladeny vysoké nároky na řízení takových procesů $\longrightarrow$ nárůst hardwarové složitosti.

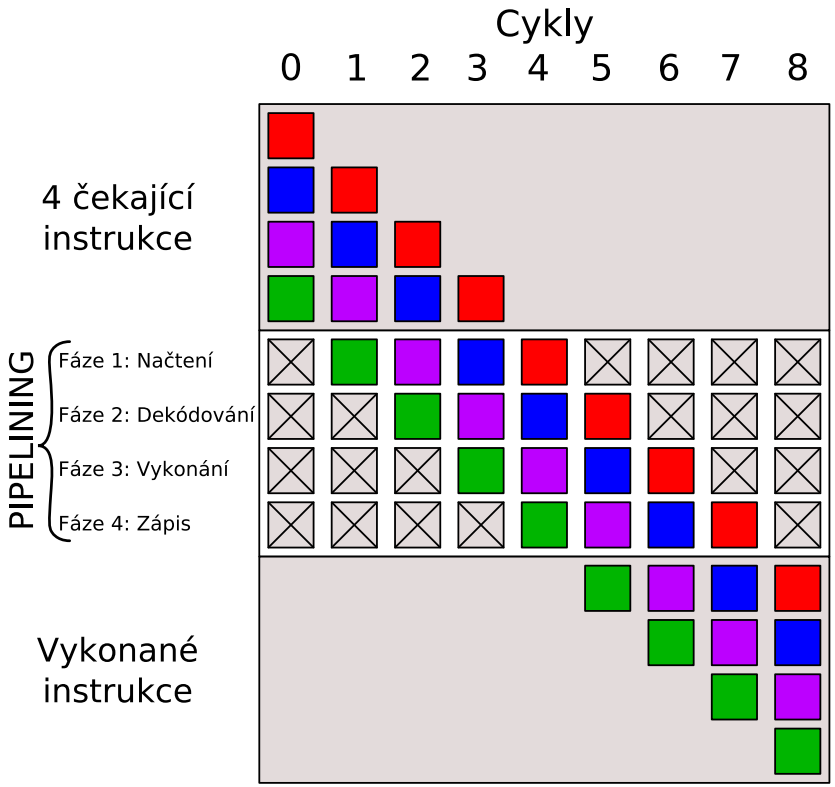
  Pozor: Nejedná se o paralelní zpracování instrukcí.
- Při klasickém zpracování instrukcí dochází ke kompletnímu výkonu jedné instrukce než se začne vykonávat instrukce druhá:
  - F_1, $D1_1$, $D2_1$, E_1, W_1.
  - F_2, $D1_2$, $D2_2$, E_2, W_2.
  - Dvě instrukce, skládající se z 5 fází, se tak vykonají za 10 strojových cyklů.
- Při zřetězeném zpracování je během 10 cyklů vykonáno 6 instrukcí:
  - Následující instrukce jsou již rozpracovány v různých stádiích.
  - Při každém dalším taktu je tak dokončena vždy jedna instrukce.
- Při zřetězení nastává problém v případě skoku v programu:
  - V tomto případě je potřeba provést vyprázdnění fronty rozpracovaných instrukcí, tzv. pipeline flush, protože výkon programu bude pokračovat na místě skoku.

Skalární/superskalární procesor

- Procesor s jedinou frontou pro zpracovávání instrukcí se nazývá skalární procesor.
- Procesor s více frontami se nazývá superskalární procesor:
  - Tato technika umožňuje procesoru vykonat více než jednu instrukci/takt.
  - Dvě fronty vykonají během 10 taktů 10 instrukcí a následující jsou rozpracovány (viz tabulka).

t_1	t_2	t_3	t_4	t_5	t_6
F_1 F_2	F_3 F_4	F_5 F_6	F_7 F_8	F_9 F_{10}	F_{11} F_{12}
	$D1_1$ $D1_2$	$D1_3$ $D1_4$	$D1_5$ $D1_6$	$D1_7$ $D1_8$	$D1_9$ $D1_{10}$
		$D2_1$ $D2_2$	$D2_3$ $D2_4$	$D2_5$ $D2_6$	$D2_7$ $D2_8$
			E_1 E_2	E_3 E_4	E_5 E_6
				W_1 W_2	W_3 W_4

Čtyř-stupňové zřetězení výkonu instrukcí



Obrázek: Čtyř-stupňové zřetězení výkonu instrukcí.

- Proces řetězení výkonu instrukcí:

Cyklus 0 4 instrukce v paměti čekající na vykonání.
1 Zelená instrukce je načtena z paměti.
2 Zelená instrukce je dekódována, fialová je načtena z paměti.
3 Zelená instrukce je vykonána, fialová je dekódována, modrá je načtena.
. . .
8 Všechny instrukce byly vykonány.

Řízení procesů, multi-tasking

- Systém umožňuje tzv. multiprogramování (multi-tasking), jestliže je schopen načíst několik procesů do paměti a rozdělit jejich činnost pro řídicí jednotku. Tj. několik procesů (programů) je provozováno "současně" na jediném procesoru (funkční jednotce).
- Proces se skládá z doby výkonu (Execution) a z doby, kdy čeká, např. na data (Waiting).

Princip: Zatímco jeden proces čeká na dat, může být druhý vykonáván.

Př.: Uvažujme dva procesy A a B, které mají následující strukturu:

A: E W E W E
B: E W E

- Bez multi-taskingu: E_A W_A E_A W_A E_A E_B W_B E_B.
- S řízením více úloh: E_A E_B E_A E_B E_A.

- Problém alokace procesoru pro dílčí procesy: jakým způsobem efektivně rozdělit jednotlivé čekající procesy.

Multi-tasking

- Existuje několik základních způsobů jak výkon procesů přerozdělovat:
  - First Come First Served.
  - Shortest Job First.
  - Dle určené priority procesu.
  - Round Robin.
- First Come First Served:
  - Pořadí výkonu procesů je totožné s pořadím požadavků na jejich vykonání.

Příklad

Nechť mají tři jednoduché procesy P_1, P_2 a P_3 potřebnou dobu výkonu 12, 2 a 2 periody hodinového signálu. Pořadí požadavků je 1, 2, 3.

Řešení

P_1 *Výkon procesu P_1 během period T_0 až T_{11}.*

P_2 T_{12} *až* T_{13}.

P_3 T_{14} *až* T_{15}.

Metody multi-taskingu

- Shortest Job First:
  - Nejprve se vykoná proces, který má specifikovanou nejkratší dobu výkonu.

Problém: Jak předem předvídat dobu výkonu procesů?
Řešení: Přidat ke každému procesu maximální povolenou dobu výkonu a tu pak považovat za samotnou dobu výkonu.
Problém: Riziko nevykonání dlouho trvajících procesů, v případě neustálých požadavků krátkých procesů na vykonání.

- Řazení procesů dle priority:
  - Ke každému procesu je asociována jeho priorita. Procesy jsou pak vykonány v pořadí podle jejich priority.

Problém: Riziko nevykonání procesů s příliš nízkou prioritou v případě neustálého příchodu důležitých procesů.
Řešení: Aplikace techniky stárnutí, tj. periodické zvyšování priority čekajících procesů.

Metody multi-taskingu

- Round Robin:
  - Celkový čas procesoru je rozdělen na "časová kvanta" (definované časové intervaly). Jednotlivé procesy se pak periodicky vykonávají po těchto intervalech.
  - Pořadí jejich výkonu je dáno příchodem požadavků na jejich výkon.

Příklad

Nechť mají tři jednoduché procesy P_1, P_2 a P_3 potřebnou dobu výkonu 12, 2 a 2 periody hodinového signálu a kvantum (interval) je roven 1 periodě hodinového signálu. Pořadí požadavků je 1, 2, 3.

Řešení

- *Přepínání mezi procesy je následující:*
  - $P_1, P_2, P_3, P_1, P_2, P_3, P_1, P_1, \ldots, P_1$.
  - *Doba výkonu všech těchto procesů je tedy 16 period hodinového signálu.*

Obsah přednášky

Zvyšování početního výkonu procesorů
- Kalíme železo, ladíme HW

High performance computing
- Trends in parallel architectures and programming
- Multicore CPU – MPI
- Multicore CPU – OpenMP
- MATLAB parallel computing

High performance parallelism with graphics processing unit
- NVIDIA's parallel computing architecture – CUDA

Multi-tasking, pipelining
- Zřetězené zpracování instrukcí, pipelining
- Multiprogramování, multi-tasking

Vyrovnávací paměť Cache

Vyrovnávací paměti Cache

- Cache paměť je ve své podstatě vyrovnávací paměť mezi pomalým a rychlým zařízením.
- Účelem cache je urychlit přístup k často používaným informacím. Pomalým zařízením může být například pevný disk a rychlým zařízením např. hlavní paměť.
- Existují dva typy cache a to softwarová a hardwarová. Softwarová cache může byt například v operačním systému pro I/O operace. Hardwarová cache je implementována přímo v hardwaru a je řešena jako rychlá statická SRAM paměť.
- U moderních procesoru se vyskytují hierarchické cache. Tyto cache bývají označené jako L1, L2, L3, atd. Kde platí pro přístupovou dobu t_{acc}: $t_{accL1} < t_{accL2} < t_{accL3} < t_{accLx} < t_{accRAM}$, kde t_{accRAM} je přístupová doba do hlavní paměti.
  - Pro moderní více jádrové procesory je L1 cache rozdělena na datovou a instrukční a pracuje na frekvenci procesoru.
  - L2 cache je společná pro data i instrukce, je o něco pomalejší než L1 cache a mívá i větší kapacitu.
  - L3 cache je společná pro všechna jádra na čipu a je nejpomalejší a mívá největší kapacitu. Připojena k hlavní pam.

Mapování cache do pamětí, asociativita

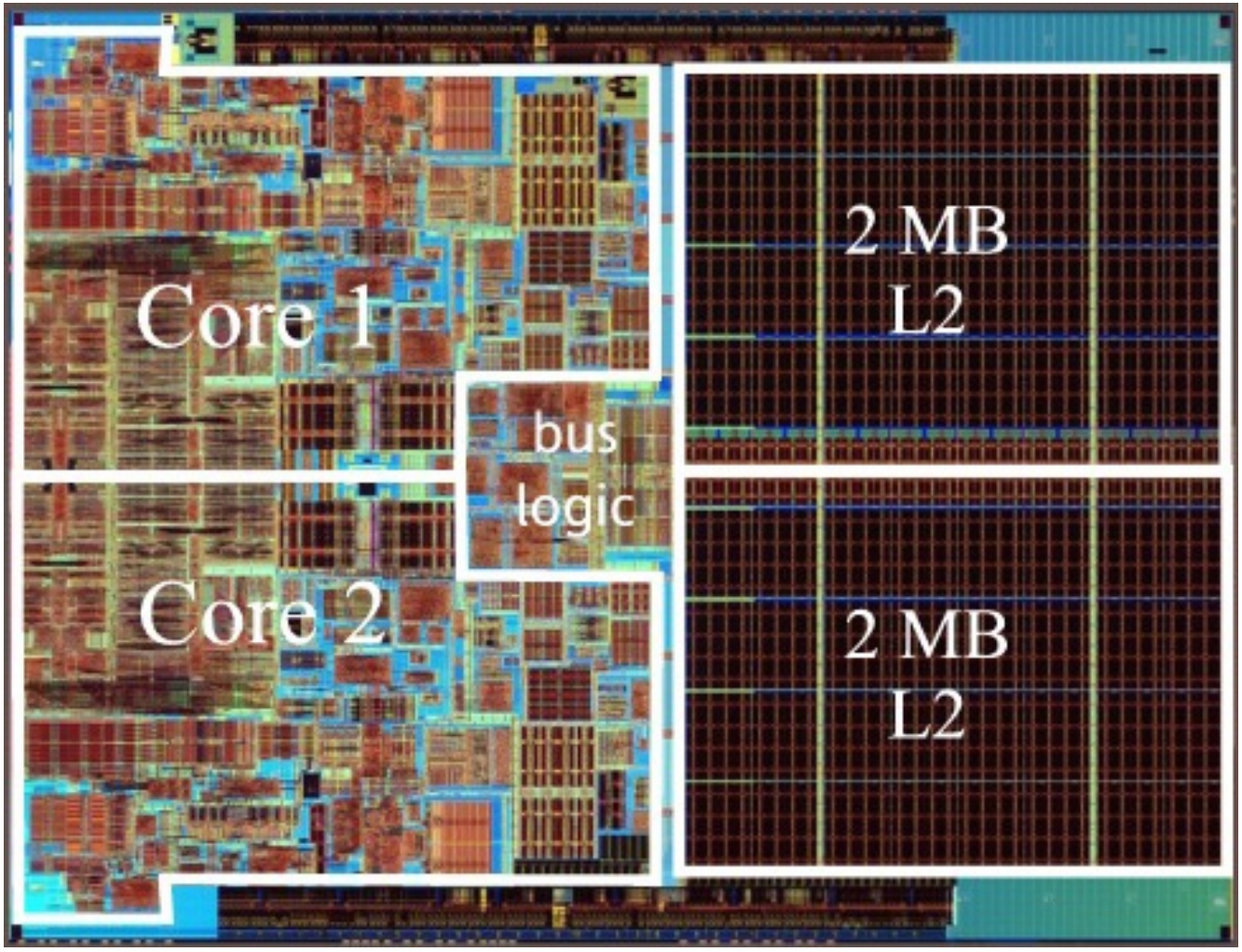
- Přímo mapovaná cache:
  - Tato cache rozdělí paměť na tolik bloků, kolik má cache linek/slotů.
  - Cache linka/slot může vypadat následovně pro 32bitový adresový prostor a cache o velikosti 512 kB rozdělíme na bloky o 32 B získáme tak 16 kB cache linek/bloků a rozdělení adresy bude vypadat následovně:

    Tag: 13 bitů, Slot: 14 bitů, Word: 5 bitů.
    Kde Word určuje bajt v cache lince/bloku, Slot je adresa cache linky/bloku a Tag určuje nejvýznamnější bity adresy, které jsou součástí cache line/bloku a je ho třeba porovnat s požadovanou adresou.
- Plně asociativní cache:
  - Jde o nejjednodušší formu cache, která v hardwarovém provedení není moc výhodná pro velké kapacity cache, protože potřebuje tolik komparátoru kolik má linek/bloků. Tato cache má u každé linky/bloku celou adresu, se kterou je třeba paralelně pro všechny cache linky/bloky při přístupu kontrolovat, abychom se dostali k požadovaným datům.
  - U této cache může v každé lince být jakákoliv adresa.
- N-cestně asociativní:
  - Tato cache je kombinací obou výše zmíněných způsobů a jedná se nejčastěji používanou cache v procesorech.
  - V tomto případě je cache rozdělena do skupin. Každá skupina obsahuje N cache linek/bloků (N většinou bývá 2, 4, 8 atd.), podobných přímo mapované cache.
  - Přístup probíhá následovně: podle Slotu/indexu se vybere skupina a porovná, podle tagu se vybere ve které lince/bloku z N jsou požadovaná data, následně se vybere požadovaný bajt dle pole Word.

Vyrovnávací paměti Cache

- Při velkých frekvencích hodinového signálu je kritická přístupová doba k operační paměti.
- Z důvodu vysokých kapacit, ceny, složitosti je používána paměť typu DRAM; má ale menší přístupové doby než SRAM.
- Aby nebyl přístup k DRAM omezující používá se vyrovnávací paměť cache
  - Paměť cache mezi procesor a paměť (L2, sekundární) pro data. Realizace pomocí rychlých SRAM. Max. kapacita jednotky MB. Původně externí součástka, nyní integrována v procesoru,
  - Interní paměť cache přímo v procesoru (L1, primární). Menší kapacita než L2 - desítky kB. Určena jak pro data, tak i instrukce,
  - Příklad: CPU Intel Core 2 1,86GHz. L1 Data Cache 32kB (2×). L1 Instruction Cache 32kB (2×). L2 Cache 2MB,
  - Cache úrovně L3 u 4jádrových procesorů - jako původní použití L2, tj. externí mezi procesorem a pamětí.

Intel Core Duo



Obrázek: Poměr mezi plochou L2 vůči zbytku procesoru Intel Code Duo.

AMD Quad-Core