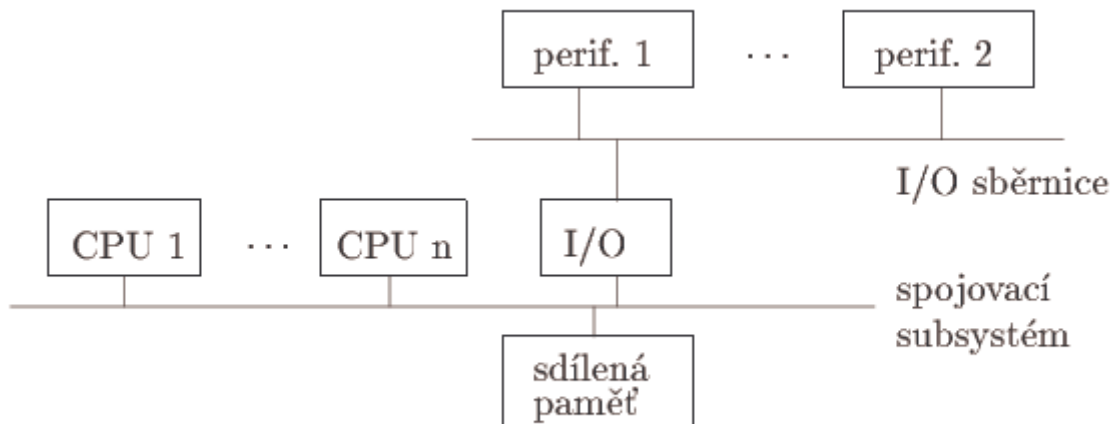


Základní pojmy

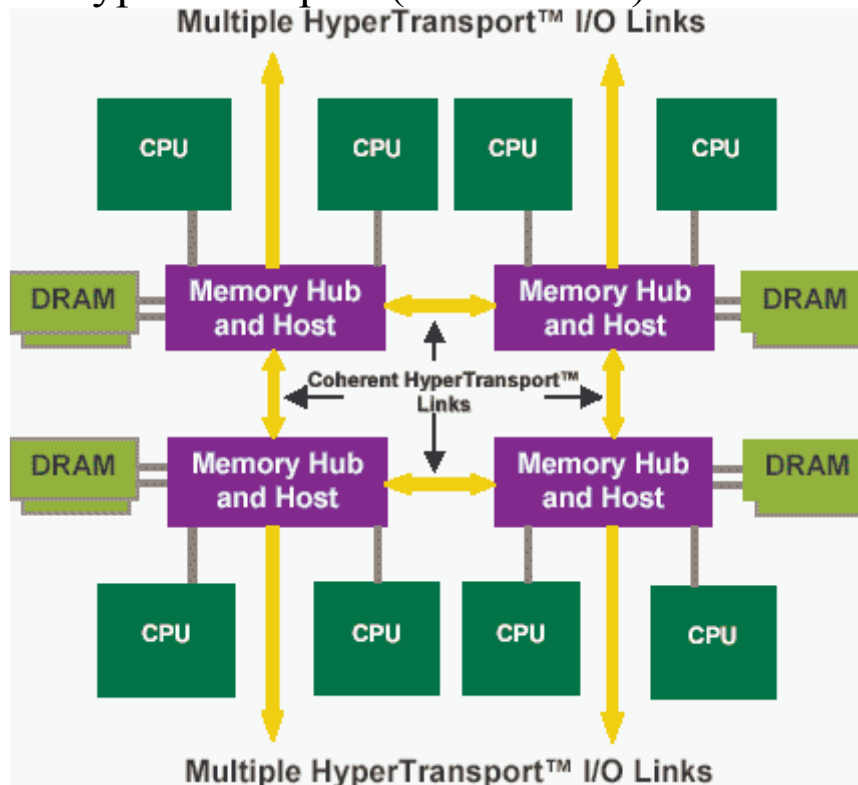
- Program, programový kód
 - statický popis výpočetního postupu, tj. co bude počítač provádět
 - nemá stav
- Assembler
 - Překladač do strojového kódu cílového procesoru
 - I když formálně nesprávně, používá se i k označení jazyka symbolických adres, ve kterém budou některé ilustrativní příklady
 - x86/x64
 - dokud neuvidíte, do čeho vlastně procesor nutíte, neznáte skutečné náklady kódu ve vyšším jazyku
- Vlákno (fiber, thread, task)
 - vykonávaný programový kód
 - tj. aktivita v čase, která má svůj stav, který je určen
 - aktuálním místem v programu (CS:RIP),
 - obsahem registrů procesoru
 - a obsahem zpracovávaných dat
 - vlákno v jednom časovém okamžiku běží pouze na jednom procesoru
- Proces
 - Dynamická kolekce vláken => má stav daný vlákny
 - Vždy obsahuje alespoň jedno vlákno, které vytvořil operační systém/interpret (např. Java)
 - Vlastní prostředky, které operační systém/interpret přidělil důsledkem činnosti některého vlákna
 - Vlákna jednoho procesu mohou běžet současně, je-li k dispozici více než jeden procesor

- Distribuovaná aplikace
 - Několik spolupracujících procesů
 - Obvykle jsou distribuovány na uzly počítačové sítě
 - Jeden uzel má n procesorů
 - V extrémním případě mohou být na jednom uzlu
 - Dříve, kdy ještě platilo 1 proces = 1 vlákno, se spouštělo několik instancí jednoho programu
- Paralelní výpočet
 - Buď vícevláknový program,
 - Nebo distribuovaná aplikace
 - Výpočet je dynamicky strukturován na procesy a vlákna
- Paralelní program
 - Kód je staticky strukturován na podprogramy vláken
 - Má deklarovaná sdílená data
- Interakce
 - Spolupráce na úrovni vláken uvnitř procesů
 - Spolupráce na úrovni procesů distribuované aplikace
 - Výměna informací
- Synchronizace
 - Forma interakce
 - Zajištění správné návaznosti operací

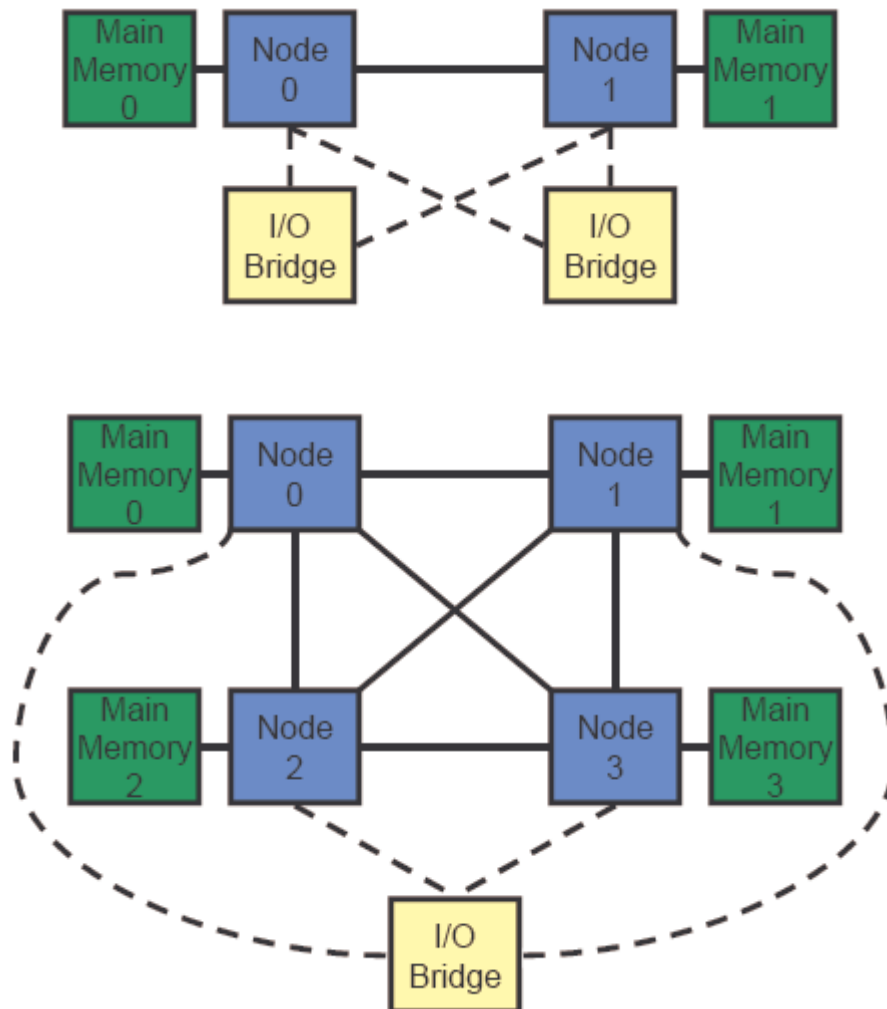
Symetrický multiprocesor (SMP)



- Všechny procesory jsou identické
- Vlákno může být alokováno na libovolný procesor
- Pro plánovač OS je to jednodušší, než kdyby se procesory významně lišily
- Procesory sdílejí jednu paměť – úzké hrdlo je sběrnice
 - Řešením je spojit jednotlivé procesy přímo – tzv. point-to-point
 - Hyper Transport (Consortium)

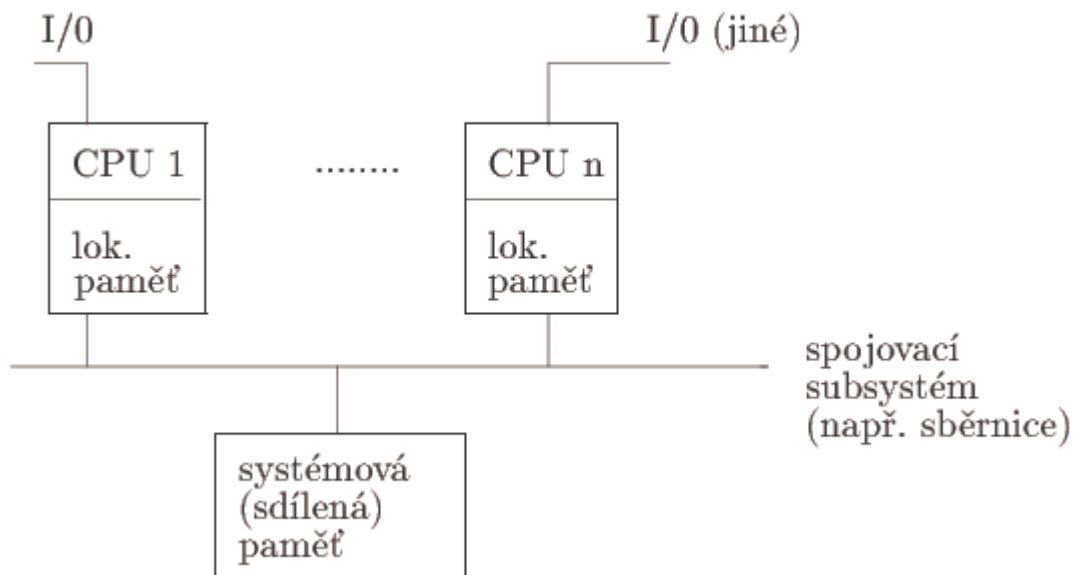


- Intel QuickPath Interconnect
 - Též známo jako Common System Interface (CSI) nebo jenom QuickPath



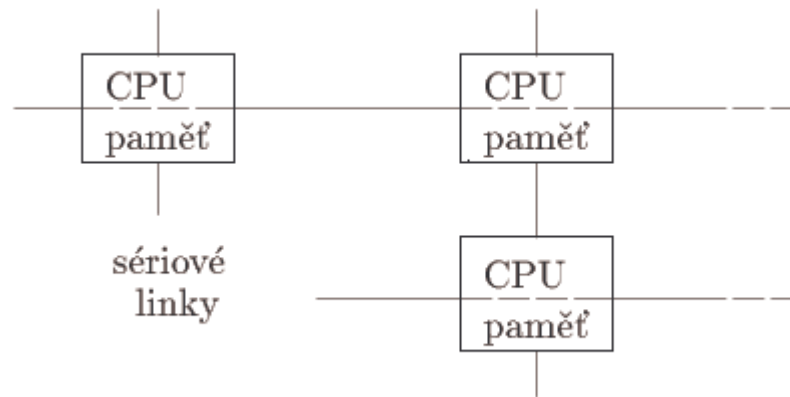
<http://www.realworldtech.com/page.cfm?ArticleID=RWT082807020032&p=8>

Asymetrický multiprocesor (ASMP)



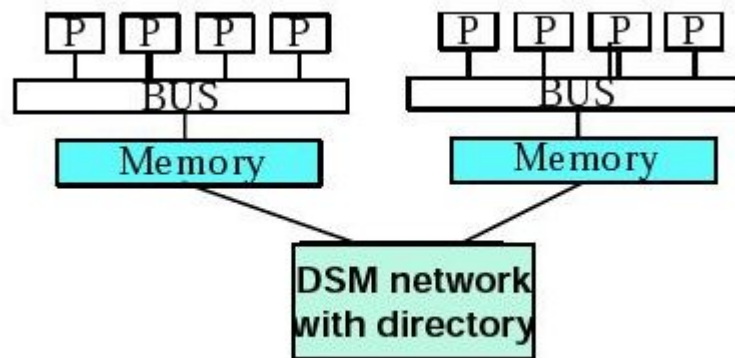
- Kromě sdílené paměti, každý procesor má vlastní lokální paměť a vlastní připojení I/O
- Každý procesor může mít jinou instrukční sadu
 - v extrémním případě na každém z nich běží jiný OS
- OS nemůže alokovat libovolný proces na libovolný procesor
- Jednotlivé procesory mohou vykonávat specifické úkoly
- Sdílená paměť může obsahovat pouze minimum dat
 - Minimalizace úzkého hrdla sběrnice
- V '80 prohrál se SMP na poli univerzálních počítačů
 - Řešení ATI, kdy jedna karta renderovala scénu a druhá se starala fyziku, je softwarový ASMP
 - Sony PS3 – každý procesor pouze pro určité úkoly

Multiprocesor s distribuovanou pamětí



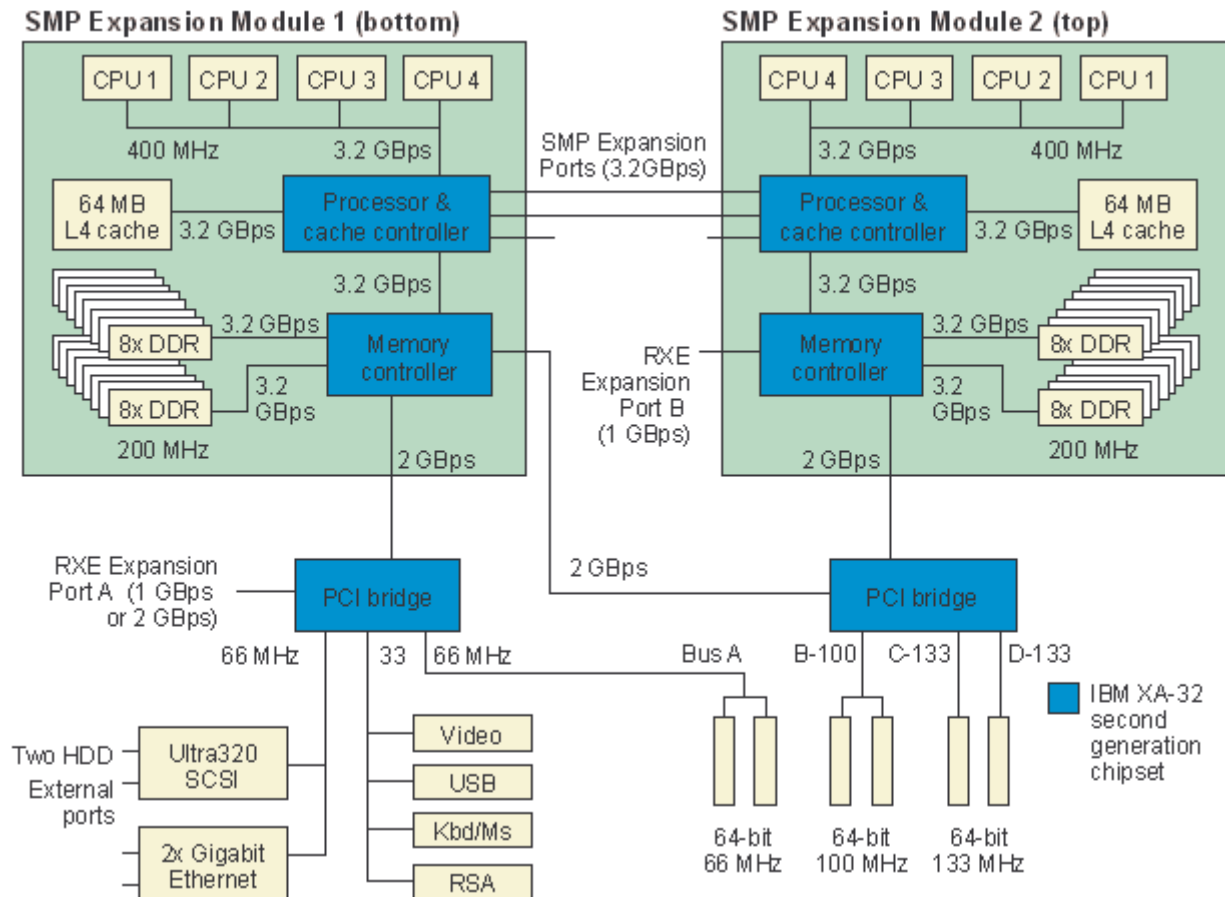
- Každý procesor má svou lokální paměť
- Sdílná paměť není
- Každý uzel je v podstatě počítač s omezeným I/O
- Komunikuje se zasíláním zpráv
- Typické topologie jsou 2D cyklicky uzavřené mřížky, nebo n-rozměrná krychle
- Výhodou je odstranění jedné sběrnice jako úzkého hrdla
- Nevýhodou je však malá univerzálnost – výkonnost záleží na způsobu alokace procesů na uzly a použitém komunikačním schématu
- Vhodné např. pro tzv. pipe-lines
- Transputery - Occam

NUMA



<http://upload.wikimedia.org/wikipedia/en/c/c1/Numa.jpg>

- Non-Uniform Memory Access
- DSM – Distributed Shared Memory
- Další pokus o vylepšení škálovatelnosti, kde jedna sběrnice SMP systémů představuje limit
 - Existuje několik sběrnic, které jsou vzájemně propojené
- Zavádí pojem lokální a vzdálené paměti
- Každý procesor má svůj „kus“ paměti
- Doba přístupu do paměti procesorem se může lišit, podle toho, kde paměť fyzicky je => non-uniform
- Problém, když proces odmigruje příliš daleko na jiný procesor
- Podporováno AMD Opteron – viz Hyper Transfer
- Podporováno Intel Nehalem a Tukwila – viz QuickPath



<http://www.redbooks.ibm.com/abstracts/tips0476.html?Open>

SUMO

- AMD Opteron
- Sufficiently Uniform Memory Organization
- Fyzicky podobné NUMA
- Pro OS se tváří jako SMP => aneb jak spustit normální jádro OS na NUMA

COMA

- Cache Only Memory Architecture
- NUMA – lokální paměti tvoří celou paměť, při vzdáleném přístupu vznikají repliky
- COMA – lokální paměti jsou použity jenom jako cache, lepší využití paměti, ale je zde problém platnosti replik
- Byly navrženy i hybridní kombinace NUMA-COMA
 - Sun WildFire

Distribuovaný systém

- De facto počítačová síť, která se může tvářit jako jeden stroj
- Dědí problémy multiprocesoru s distribuovanou pamětí
- Každý uzel je plnohodnotný počítač, na kterém může běžet několik procesů zároveň
- Např. cluster
 - Počítače propojené velmi rychlou lokální sítí

Vektorový paralelní počítač

- Vector Processor, Array Processor
- Mohou provádět operace s vektory čísel na úrovni instrukcí strojového kódu
- Dříve doména superpočítačů jako Cray-1, dnes
 - MMX (Intel) – Matrix Math eXtension?
 - SSE1-5 (Intel, AMD) – Streaming SIMD Extensions
 - 3DNow! (AMD)
 - AVX (Intel) – Advanced Vector Extensions
 - AltiVec (IBM, Apple, Motorola)
- Paralelizaci tohoto typu využívá překladač
 - Některé (např. GCC v4) umí tzv. auto-vektORIZACI, kdy je část kódu rovnou převedena na vektorové instrukce
 - Programátor může překladači pomoci správným zápisem kódu

```
procedure VectorAdd (dst, src:PVector); assembler;
asm
...
    mov ecx, VectorSize    //128 bits = 16 bytes => 4
@loop:

    fld    [eax]    //dst -> ST0
    fadd   [edx]    //src
    fst    [eax]    //ST0 -> dst
    add    eax, 4
    add    edx, 4

    dec    ecx
    cmp    ecx, 0
    jne    @loop
...
end;
```

type

```
PVector = ^TVector;
TVector = packed record
    x, y, z, w: single;
    //sizeof(single) = 4
end;
```

SSE

```
movups    xmm0, [eax] //dst
movups    xmm1, [edx] //src
addps     xmm0, xmm1
movups    [eax], xmm0 //dst
```

Základní modely paralelní dekompozice

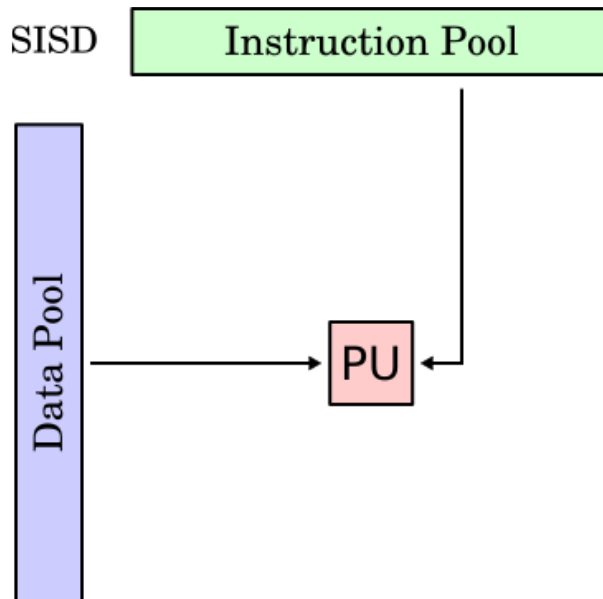
- Vlastnímu programování předchází fáze analýzy!
- Dekompozice výpočtu na paralelizovatelné složky
- Návrh, co bude řešit ten který proces/vlákno
- Dekompozice dat pro jednotlivé procesy/vlákna
- Obecně rozhodování nemusí být ovlivněno cílovou architekturou
 - Uvedené modely jsou dostatečně obecné
 - Nicméně, lokální a distribuovaná paměť představují dost zásadní rozdíl, než aby se nevzal do úvahy
 - Analogie lze nalézt/postupy lze uplatnit i v běžném životě (je-li to politicky průchodné:-)

Flynnova taxonomie

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- Uvedené jsou čtyři základní klasifikace podle Flynna
 - Počet konkurenčně prováděných instrukcí
 - tj. programů a někdy používanému počestěnému označování např. SPMD, což ale není formálně správně právě podle Flynnna – viz dále
 - Počet existujících dat (Data Streams)
- MIMD – dále se dělí na
 - SPMD – Single Program, Multiple Data Streams
 - MPMD – Multiple Program, Multiple Data Streams
- Používají se termíny funkční a datový paralelismus

SISD



- Single Instruction, Single Data Stream
- Sekvenční výpočet, žádný paralelismus
- Např. i386 s MS-DOSem

<http://en.wikipedia.org/wiki/Image:SISD.svg>

```
COsoba princ, princezna, kral;
CPotvora drak;
CObjekt kralovstvi;
```

```
HRESULT __fairytale Edvenčra() {

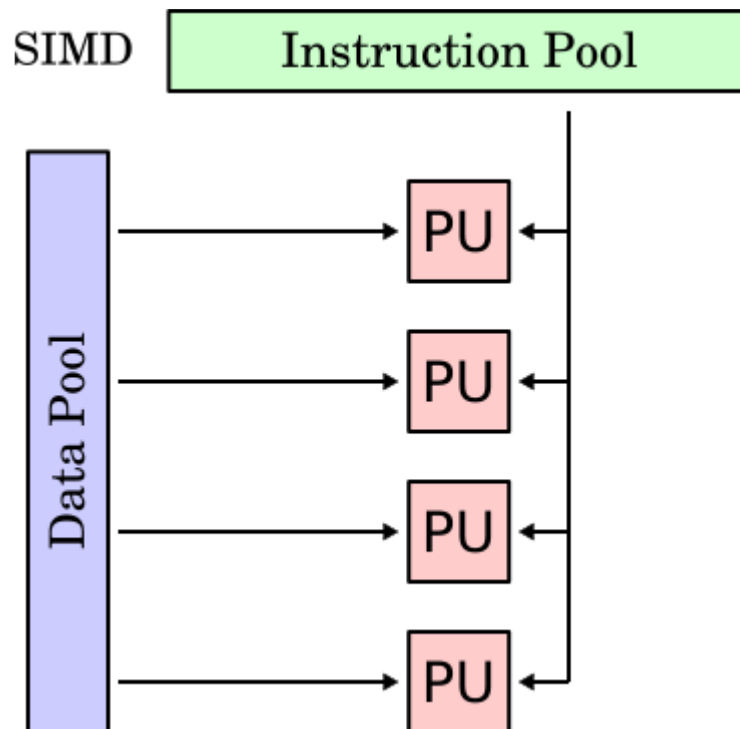
    if (princ.sila > drak.sila) {
        delete drak;
        princ += princezna + kralovstvi/2;
        return S_OK;
    }

    else if (princ.sila < drak.sila) {
        princ -= hlava;
        drak.hmotnost += princezna.hmotnost;
        kral.status = "Smutny";
        return E_FAIL;
    }

    else if (princ.sila == drak.sila) {
        princ.perform("BuyBiggerGun");
        return S_ToBeContinued;
    }

}
```

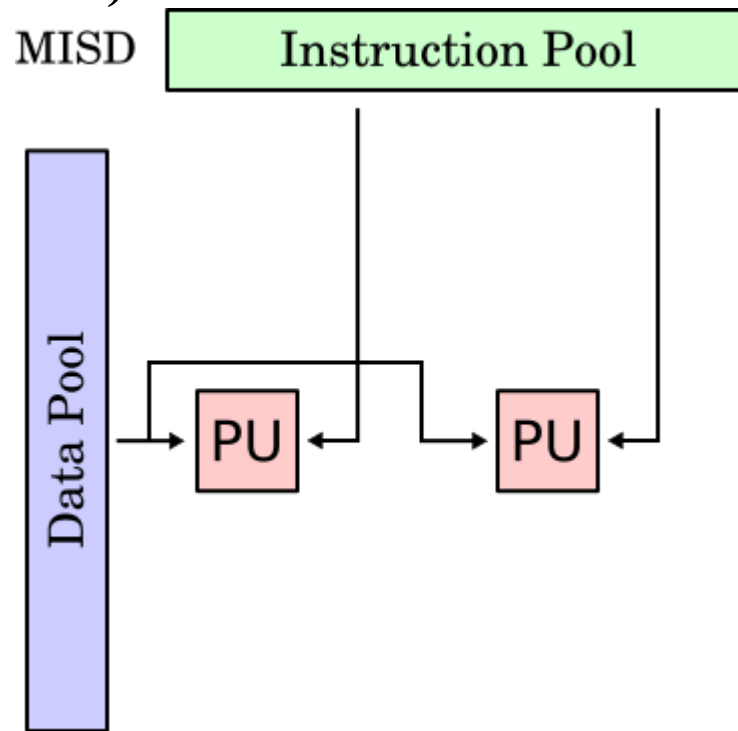
SIMD



<http://en.wikipedia.org/wiki/Image:SIMD.svg>

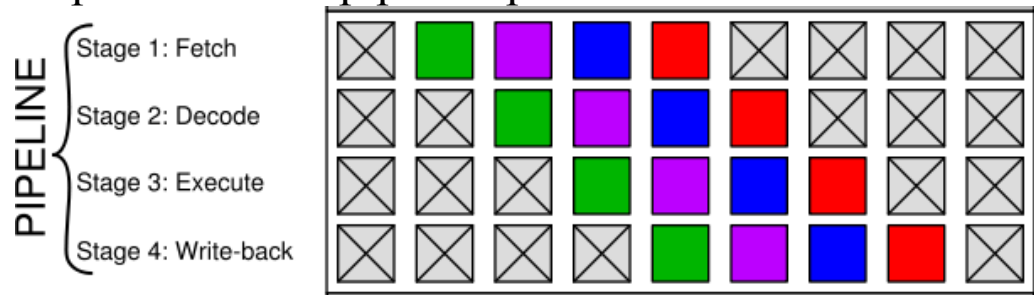
- Single Instruction, Multiple Data Streams
- Viz výše vektorový paralelní počítač

MISD (MPSD)



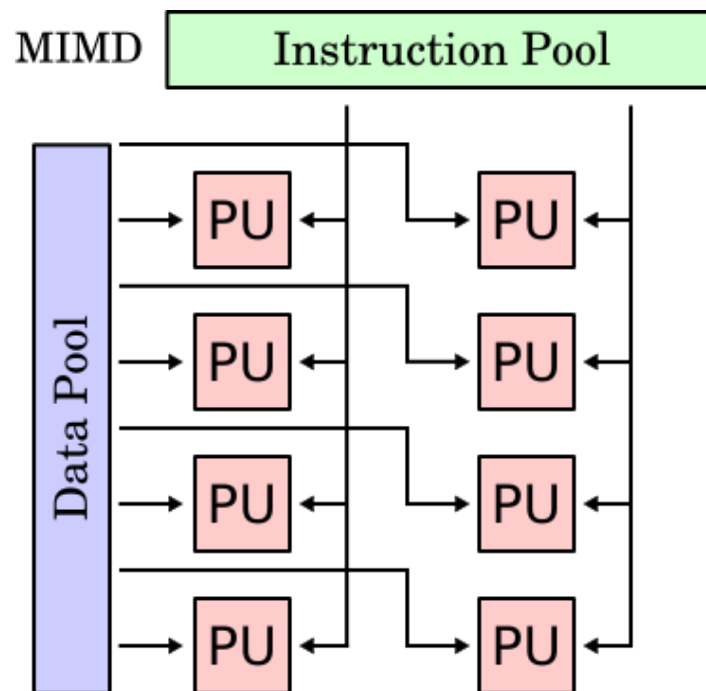
<http://en.wikipedia.org/wiki/Image:MISD.svg>

- Multiple Instruction (Program), Single Data Stream
- Používáno pro výpočty odolné proti poruchám (Fault Tolerant)
 - Několik různých systémů zpracovává ty samá data a musejí se shodnout na výsledku – např. řízení letu raketoplánu, letadla, atd.
- Používá se v tzv. Pipeline architektuře
 - několik procesů zpracovává data v jednom datovém proudu
 - analogií je montážní linka v továrně
 - Např. instrukční pipeline procesoru



http://en.wikipedia.org/wiki/Image:Pipeline%2C_4_stage.svg

MIMD



<http://en.wikipedia.org/wiki/Image:MIMD.svg>

- Multiple Instruction, Multiple Data Streams
- Několik procesorů zároveň vykonává různé instrukce nad několika různými daty
- Procesory pracují asynchronně a nezávisle na sobě
- Procesory buď mají sdílenou paměť,
 - Programátorovi snáze srozumitelný přístup
 - O zajištění integrity dat se stará OS
- nebo distribuovanou
 - má lepší škálovatelnost
- Distribuovaný systém – viz výše i dále
- MIMD programy lze odladit i na jednom počítači
 - Virtualizace jednoho procesoru pro sdílenou paměť
 - localhost a pro distribuovanou paměť
 - Samozřejmě, některé chyby takto mohou uniknout, protože se neprojeví díky jinému komunikačnímu zpoždění, nebo díky virtualizaci jednoho procesoru, kdy žádná dvě vlákna neběží současně

SPMD

- Single Program, Multiple Data Streams
- Několik procesorů autonomně vykonává jeden program nad různými daty
- Bod vykonávání programu nemusí být na všech procesorech stejný
- Označuje se též jako dekompozice dat
- Používá se ke zpracovávání velkých objemů dat
 - k procesů běžících podle stejného programu zpracovává strukturně stejná, ale hodnotově různé části dat
 - např. násobení matic – každý prvek, řádek, či sloupec matice lze spočítat jedním procesem/vláknem
- Sleduje se čistě výkonnostní hledisko
- Předpokládá se, že každý z procesorů je schopen vykonávat ten samý program

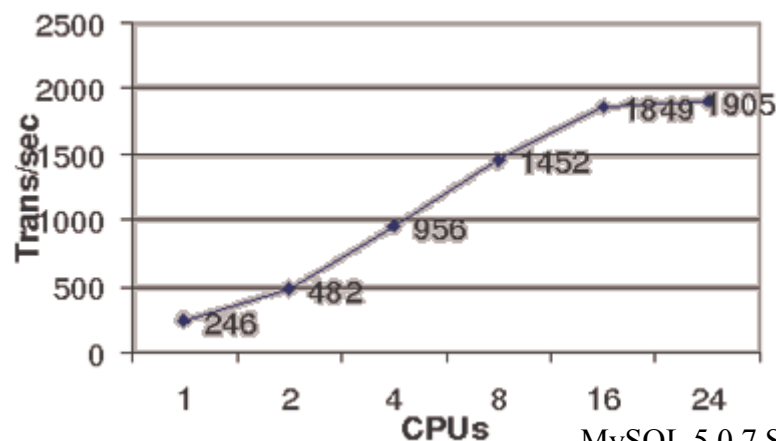
MPMD

- Multiple Program, Multiple Data Streams
- Několik procesorů autonomně vykonává více než jeden program nad různými daty
- Např. farmer-worker, kdy jeden proces úkoluje ostatní
- Nemusí jít nutně pouze o urychlení výpočtu
 - Může jít o aplikaci, kdy se každý proces stará „o to svoje“ a zároveň spolupracuje s ostatními
 - Např. Cisco IOS – síť A s OSPF, síť B s EIGRP, border router vyplní směrovací tabulku podle obou
- Dist. Simulace – systém spolupracujících komponent

Interakce

- Po výběru modelu dekompozice výpočtu je třeba stanovit model interakce procesů
- Rozhodnutí je výrazně ovlivněno
 - architekturou systému, který máme k dispozici,
 - distribuovaná vs. sdílená paměť
 - prostředky poskytovanými OS, nebo interpretem
 - WinAPI vs. POSIX vs. Java
 - a použitými knihovnami funkcí
 - PVM vs. MPI vs. využití programovatelné sítě
- Interakce zahrnuje synchronizaci a výměnu dat
 - Žádoucí je
 - Minimální interakce
 - Rychlostní nezávislost výpočtu
 - škálovatelnost
- Minimální interakce
 - Co největší samostatnost procesů
 - Častá komunikace zatěžuje komunikační kanál (sběrnice, Ethernet) a tím zpomaluje celý systém
 - Tedy i samotný výpočet
 - Communication overhead
 - Čím řidší komunikace, tím snáze se odhalí chyby vzniklé chybnou interakcí procesů
 - Při použití blokujících funkcí, zejména receive, dochází u časté komunikace k výraznému zpomalení příjemce
 - Totálně asynchronní paralelní algoritmus – minimální potřeba interakce procesů

- Rychlostní nezávislost výpočtu
 - Obecně nelze dopředu stanovit rychlost, s jakou poběží vlákna procesů
 - Počet běžících vláken na daném procesoru
 - I/O operace
 - Heterogenní hardware v síti
 - Paralelní výpočet musí skončit správným výsledkem bez ohledu na to, jakou rychlostí běží vlákna jeho procesů
 - V programu se definují synchronizační body, kdy se čeká až požadovaný počet ostatních vláken bude v požadovaném stavu
 - Odeslání a přijetí zprávy
 - Bariéra
 - P/V (KIV/ZOS)
 - Nedojde k zablokování výpočtu (deadlock, livelock)
 - Procesy a jejich vlákna skončí za předem definovaných podmínek
 - K modelování lze použít Petriho sítě (KIV/VSP)
- Škálovatelnost
 - Nezávislost na počtu dostupných procesorů
 - S rostoucím počtem procesů narůstá i komunikační režie a dochází ke zpomalování výpočtu



MySQL 5.0.7 SysBench CPU Scalability Test
http://developers.sun.com/solaris/articles/mysql_perf_tune.html

Komunikace

- Sdílení dat
 - Realizovatelné pouze na počítačích se sdílenou pamětí
 - Procesy sdílejí popis datových struktur
 - Procesy se nemusejí vzájemně znát, ale musí vědět, kde jsou uložena data
 - Prostředky pro přístup k datům jim dává OS
 - Může dojít ke konfliktu při přístupu k datům
 - Jeden proces je může číst
 - Jiný proces je bude zapisovat
 - Příslušný kód programu se nazývá kritická sekce
 - K jejímu zabezpečení se používají synchronizační primitiva
 - Nízkoúrovňové: semaforey, zámky, bariéry
 - Vysokoúrovňové: monitory, rendez-vous
- Zasílání zpráv
 - Jediná možná v systémech s distribuovanou pamětí
 - Lze ji použít i v systémech se sdílenou pamětí
 - WinAPI: SendMessage
 - Univerzální technika
 - Ne, když se použijí funkce specifické pro jednu cílovou architekturu
 - Je pomalejší než sdílení dat
 - Synchronní – blokující
 - zpráva je předána, až jsou na to oba připraveni
 - Asynchronní – neblokující
 - Odesílatel ji odešle, vloží do vyrovnávací paměti a příjemce si ji vyzvedne později

- Symetrické – zpráva obsahuje adresu odesílajícího i příjemce
- Asymetrické – zpráva obsahuje jen adresu příjemce
- Nepřímé – zpráva obsahuje adresu komunikačního kanálu, tj. komunikující procesy se vůbec nemusí nijak znát

Výkonnost

- Složitost
 - Complexity, worst-case complexity
 - Maximální doba výpočtu algoritmu pro všechny možné kombinace vstupních dat
 - $O(n) = n$
 - složitost sekvenčního algoritmu, která je lineárně závislá na n počtu prvků
 - např. součet čísel
 - KIV/PPA2
 - Paralelní součet na $p = n/2$ procesorech má složitost $O(\log n)$
 - Logaritmus s libovolným základem
- Cena (paralelního algoritmu)
 - Cost
 - Složitost vynásobená počtem procesorů
 - Např. $(\log n) * n/2$
 - Je úměrná celkovému strojovému času všech použitých procesorů

- Urychlení
 - Speedup
 - Existuje několik způsobů, jak ho vyjádřit, následující je nejrozšířenější/nejznámější způsob
 - Poměr doby výpočtu referenčního algoritmu a porovnávaného algoritmu
 - Např. nejlepšího známého sekvenčního algoritmu a paralelního algoritmu na témže (paralelním) počítači
 - $S(p) = E(1) / E(p)$
 - >1 znamená urychlení
 - Perfektní urychlení
 - Poměr je přesně roven počtu procesorů
 - Asi těžko ho dosáhnete :-)
- Účinnost
 - Efficiency
 - Urychlení dělené počtem procesorů
 - Uvažujeme urychlení proti sekvenčnímu algoritmu
 - Sekvenční výpočet trvá 10s, paralelní algoritmus na 4 procesorech trvá 5s
 - Urychlení: 2
 - Účinnost: 0,5

- Amdahlův zákon

- Nelze dosáhnout perfektního urychlení, protože vždy bude nějaká část výpočtu provedena sekvenčně

- G, H – čas strávený sekvenčně provedeným výpočtem

- G

- čas strávený vykonáváním neparalelizovatelného kódu, tj. sekvenčně
- Unavoidably Serial

- H

- čas strávený vykonáváním paralelizovaného kódu, ale sekvenčně
- Serialized-Parallel

- $E(p) = G + H/p$

- $S(p) = (G+H)/(G+H/p)$

- Pro velké p platí $S(p) = 1 + H/G$

- To by znamenalo, že dosažení perfektního urychlení je možné, pokud se můžeme vyhnout kódu, který nelze paralelizovat

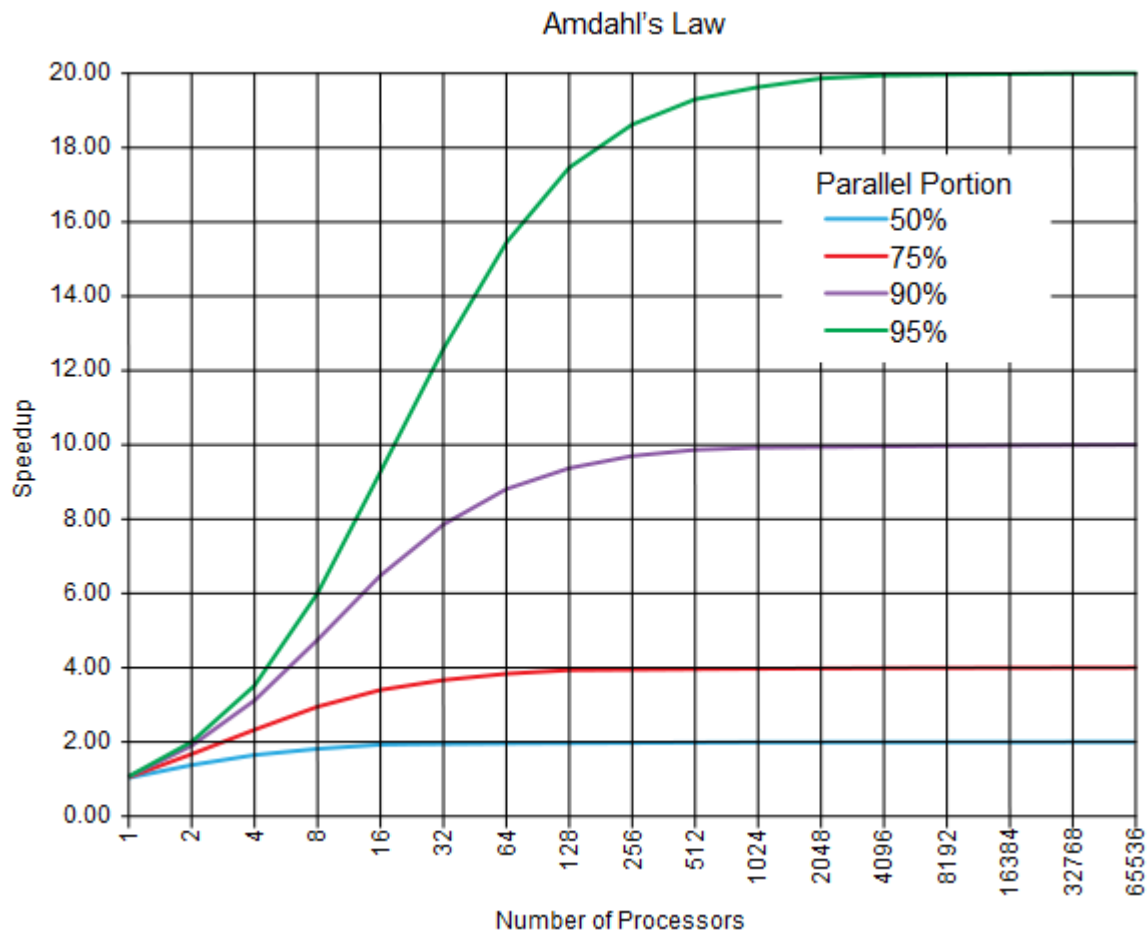
- Má vliv na H

- Ale – čím větší počet procesorů, tím např. větší režie jejich komunikace => takže přece jenom nepůjde...

- Režie OS (plánování, I/O, atd.) se o to také postará

- Lze ho také zapsat s f
(sekvenčně prováděná část kódu)

$$S \leq \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}$$



<http://upload.wikimedia.org/wikipedia/en/e/ea/AmdahlsLaw.svg>

- Anomální urychlení
 - Distribuce rozsáhlých dat u distribuované aplikace může omezit nutnost stránkovat RAM
 - S dostatečně rychlými komunikačními kanály pak dojde k rychlejšímu vykonávání programového kódu, protože odpadá čekání na zpomalující I/O operace provázející stránkování, včetně obsluhy příslušných přerušení (KIV/OS)
 - Např. paralelizované vyhledávací algoritmy mohou mít větší než lineární urychlení
 - Lze rychleji upřesnit výběrová kritéria

- Superlineární urychlení
 - Nicméně, v některých případech je skutečně možné, aby S vyšlo lépe než je perfektní urychlení
 - V těchto případech se nejedná o chybu ve výpočtu
 - Zejména, porovnáváme-li oproti nejlepšímu sekvenčnímu algoritmu
 - Cache procesoru
 - Mnohem častější cache-hit než je obvyklé
 - Výpočet je pak prováděn mnohem rychleji, než když se programový kód dostává k procesoru z pomalejší paměti
 - Záleží na konkrétním programovém kódu, zda se ho bude dostatečné množství nalézat právě v lokálních cache jednotlivých procesorů
 - Analogicky datová čachr

Výpočetní čas

- Nejjednodušeji čas strávený výpočtem a čas strávený čekáním
- Výpočetní čas procesoru
 - Díky spekulativním snahám procesorů o urychlování vykonávání kódu (řetězení instrukcí, vykonávání mimo pořadí, atd.), sběrnici, atd. je třeba uvažovat čas všech procesorů

- **Komunikační čas**
 - Zpoždění, které vznikne při interakci procesů a jejich vláken
- **Další časové prodlevy**
 - Nedostatek práce
 - Nedostupnost dat
 - Např. čekání na I/O
 - Režie OS
 - Nicméně, velké časové prodlevy také mohou znamenat nevhodný algoritmus, který je způsobuje svou činností

Testy výkonnosti

- Testování jednotlivých komponent systému dohromady i zvlášť
- Několikrát po sobě se provede specifická sekvence operací a vyhodnotí se doba trvání
- Je třeba si být vědom efektů jako je např. vliv velikost cache na vykonávání kódu, spekulativní vykonávání kódu, atd.
- **Průmyslové standardy**
 - Standard Performance Evaluation Corporation (SPEC)
 - Transaction Processing Performance Council (TPC)
- **OpenSource**
 - Dhrystone – celočíselná aritmetika
 - Whetstone – čísla s plovoucí čárkou
 - Linpack/LAPACK – numerické výpočty

Gustafsonův zákon

- $S(P) = P - \alpha \cdot (P-1)$
 - S – urychlení
 - P – počet procesů
 - α - část procesu, kterou nelze paralelizovat
- Co když budeme mít k dispozici hodně procesorů a výpočet složitý tak, že sériově prováděná část bude zanedbatelná?
 - Amdahl si s tím moc neporadí...
- Nerozdělíme výpočet podle paralelizovatelnosti kódu, ale podle podílů času – sériově vs. paralelně
- $a(n) + b(n) = 1$
 - 1 – čas výpočtu na p procesorech
 - a(n) – sériově
 - b(n) – paralelně
- $a(n) + p \cdot b(n)$
 - čas výpočtu na jednom procesoru
- $S = \frac{a(n) + p \cdot b(n)}{a(n) + b(n)}$
- Ideální urychlení
 - $p \rightarrow P$
 - $a(n)$ aka $\alpha = \lim_{x \rightarrow 0+} x$

Karp-Flattova metrika

- $$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$
 - e – metrika, podíl, kolik kódu se provedlo sériově
 - p – počet procesorů
 - ψ - urychlení na p procesorech
 - Určí se z naměřených časů

- $$T(p) = T_s + \frac{T_p}{p}$$
 - Čas výpočtu na p procesorech
 - T_s – čas po který kód běžel sériově
 - T_p – čas po který kód běžel paralelně

- $$T(1) = T_s + T_p$$

- $$e = \frac{T_s}{T(1)}$$
 - Dosadí se do $T(p)$

- $$\psi = \frac{T(1)}{T(p)}$$

- $$\frac{1}{\psi} = e + \frac{1-e}{p}$$

- A úpravou dostaneme e