Multicore
Multiprocesoare
Cluster-e

O mare perioadă de timp, creearea de calculatoare puternice ⇔ conectarea mai multor calculatoare de putere mică.

Trebuie creat software care să știe să lucreze cu un număr variabil de procesoare

Anumite operații se pot executa chiar dacă un procesor se defectează

Performanță mare ⇔ throughput mare pentru task-urile independente => paralelism la nivel de proces

Procesarea paralelă a unui program – un program care rulează pe mai multe Procesoare simultan.

Cluster – este format din mai multe microprocesoare aflate în server-e sau calculatoare independente.

Microprocesoare multicore – un microprocesor care conține mai multe procesoare (core-uri) într-un singur circuit integrat.

Programatorii în MATLAB scriu un program de înmulțire a două matrici gândind într-un mod secvențial, dar execuția se va face serial – Pentium 4 sau în paralel – Clovertown (Intel Xeon e5345)

Dificultatea cu paralelismul NU provine de la hardware.

Programele de procesare paralelă sunt mult mai greu de realizat decât cele care folosesc procesarea secvențială.

Vom avea nevoie de programarea proceselor, optimizarea încărcării procesorului, timpul pentru sincronizare și overhead-ul pentru comunicare.

Scalare puternică – creșterea de viteză atinsă pe un multiprocesor fără a crește dimensiunea problemei

Scalare slabă – dimensiunea programului crește proporțional cu creșterea numărului de procesoare.

Pentru a ușura sarcina programatorilor în cazul hardware-ului paralel s-a decis implementarea unui singur spațiu fizic de adrese care poate fi împărțit între toate procesoarele.

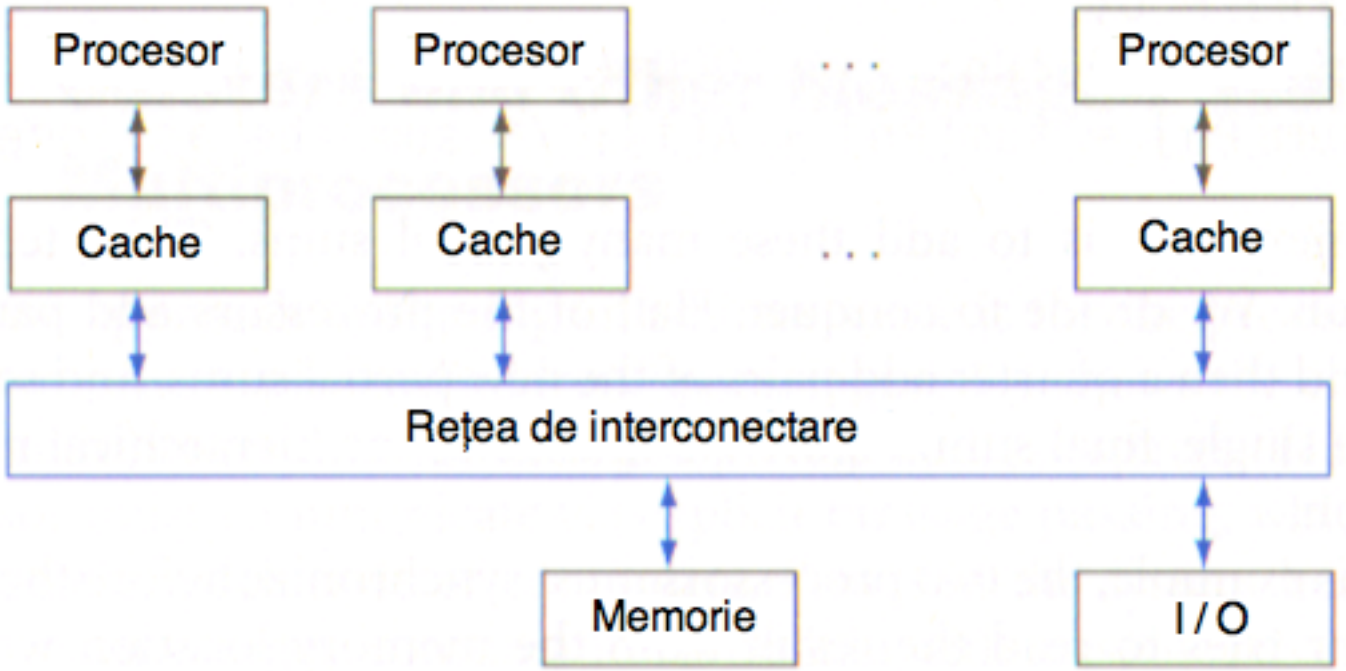
Ideea a fost ca să existe disponibilitatea variabilelor la un moment de timp pentru fiecare procesor.

Un spațiu fizic de adrese comun – chip-urile multicore – atunci va trebui asigurată corența cache-ului pentru a avea o consistență a memoriei shared.

SMP – SHARED MEMORY MULTIPROCESSOR – un procesor paralel cu un sigur spațiu de adrese, implicând comunicare implicită cu încărcările și memorările

Procesoarele comunică prin intermediul variabilelor comune în memorie.

Toate procesoarele sunt capabile de a accesa orice locație de memorie prin intermediul instrucțiunilor de încărcare/memorare.



Multiprocesoare cu un singur spațiu de adrese:

1. UMA – Uniform Memory Access – accesarea memoriei principale durează același interval de timp pentru orice procesor și pentru orice cuvânt

2. NUMA – Nonuniform Memory Access – anumite accese de memorie sunt mai rapide decât altele. Contează procesorul care întreabă dar și cuvântul solicitat

Sincronizarea – procesul de coordonare a comportamentului a două sau mai multe procese care pot rula pe procesoare diferite.

Când avem partajarea cu un singur spațiu de adrese, trebuie să avem un mecanism separat de sincronizare

Exemplu – *dorim să realizăm suma a 100.000 de numere pe o mașină de tipul SMP. Presupunem că avem 100 procesoare.*

Pasul 1: Împărțim mulțimea de numere în submulțimi de aceeași mărime.

Vom oferi diferite adrese de start pentru fiecare procesor

Toate procesoarele pornesc programul prin rularea unui ciclu care însumează propria submulțime de numere.

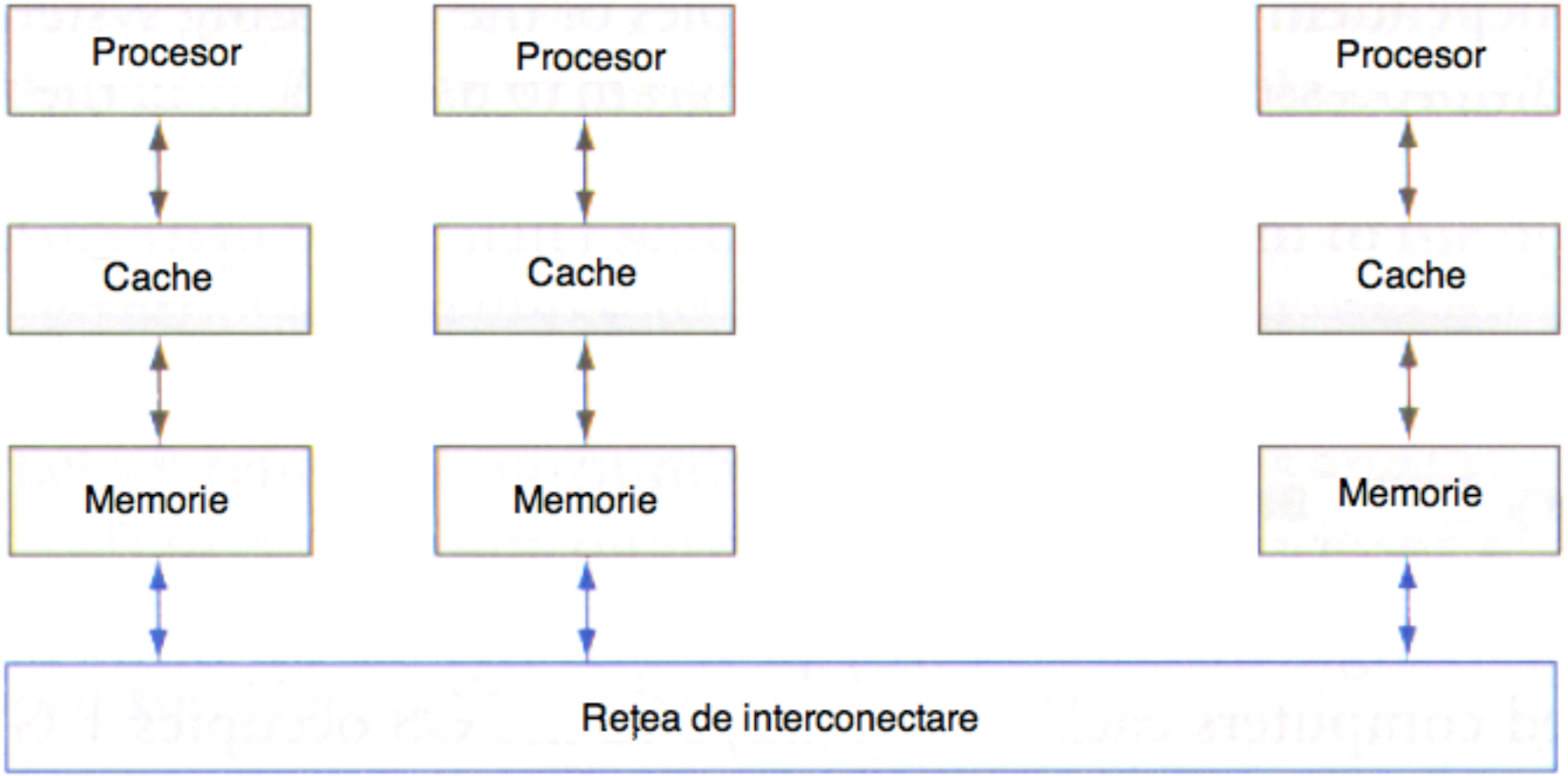
```
sum [Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
        sum[Pn] = sum[Pn] + A[i]
```

Pasul 2: Adunăm aceste sume parțiale. Acest pas este denumit REDUCȚIE

Jumătate dintre procesoare adună perechi de sume parțiale, apoi jumătate vor aduna etc

```
half = 100;           /* numărul de procesoare; i și half sunt private
repeat
        synch(); /* așteptăm pentru suma parțială
        if (half%2 !=0 && Pn == 0)
                sum[0] = sum[0] + sum[half-1];
        half = half / 2;
        if (Pn < half)
                sum[Pn] = sum[Pn] + sum[Pn + half];
until (half == 1);    /* ieșim cu suma finală în sum[0] */
```

Cluster-e



Message-passing multiprocessor: fiecare procesor are propiul său spațiu privat de adrese.

Multiprocesorul trebuie să comunice explicit prin

pasarea mesajelor – comunicarea între mai multe procesoare prin trimiterea/primirea explicită a informației

Sistemul are rutine pentru trimiterea/primirea mesajelor – procesorul știe când un mesaj este trimis. Există și mesaje de confirmare implementate.

Cele mai eficiente cluster-e sunt cele construite într-o rețea locală. Această soluție este însă foarte costisitoare.

Un mare neajuns al acestei soluții este legat de costul administrării a unui cluster. Dacă avem *n* mașini într-un cluster, administrarea trebuie făcută ca și pentru *n* mașini independente.

Cluster-ele formate din mașini virtuale sunt mult mai populare deoarece administrarea lor este mult mai facilă.

Un alt dezavantaj este constituit din faptul că mașinile din cluster sunt legate între ele prin echipamente I/O.

Un alt dezavantaj este constituit din faptul că un cluster format din *n* mașini are *n* memorii independente și *n* copii ale sistemului de operare.

Exemplu: Presupunem un procesor cu memorie partajată de capacitate 20GB – memoria principală. 5 calculatoare într-un cluster fiecare având 4GB din care OS-ul ocupă 1GB. Cât de mult spațiu este pentru utilizatorii cu memoria partajată ?

Soluție:

Raportul dintre memoria disponibilă pentru programele utilizatorilor pe un calculator cu memorie partajată versus cluster este:

(20 - 1) / 5 * (4 - 1) = 19 / 15 = 1.25 => calculatoarele cu memorie

partajată au mai mult spațiu cu 25%

Exemplu: Dorim să executăm suma a 100.000 de numere cu ajutorul unui multiprocesor cu pasare de mesaje care este format din 100 procesoare. Fiecare procesor are mai multe memorii private

Soluție: În prima fază, vom distribui 100 de subseturi către fiecare memorie locală

Determinăm suma pentru fiecare subset. Acest pas este un simplu ciclu:

```
sum = 0;
for (i=0; i<1000; i=i+1)          /*avem un ciclu pentru fiecare array
        sum = sum + AN[i];        /*suma pentru array-urilor locale
```

Trebuie realizată REDUCEREA, care adună aceste 100 de sume parțiale

Folosim din nou divide and conquer pentru a calcula suma finală.

```
limit = 100;

half = 100; /*100 procesoare

repeat

        half = (half + 1) / 2;

        if (Pn >= half && Pn < limit)   send(Pn – half, sum);

        if (Pn < (limit/2))   sum = sum + receive();

        limit = half;         /* limita superioară de send-eri

until (half == 1);
```

Hardware Multithreading

Permite multiplelor thread-uri să folosească în comun unitățile funcționale ale unui procesor.

Permite multiplelor thread-uri să folosească în comun unitățile funcționale ale unui procesor.

Pentru folosirea în comun a unităților funcționale este necesar ca procesorul să duplice starea independentă pentru fiecare thread

Fiecare thread va avea o copie separată a fișierului de registre și a PC-ului

Memoria poate fi partajată prin mecanismul de memorie virtuală care suportă multiprogramarea.

Hardware-ul trebuie să aibă abilitatea de a de schimba către diferite thread-uri foarte repede.

Hardware multithread-ing – metode de realizare

O metodă este denumită **granularitate fină**: un hardware multithreading care realizează switch-ul între thread-uri după fiecare instrucțiune.

În această situație, procesorul trebuie să facă switch între thread-uri la fiecare ciclu de ceas.

Ca și avantaj este faptul că se poate ascunde pierderea throughput-ului – instrucțiunile de la alte thread-uri pot fi executate când un thread stă.

Ca și dezavantaj – execuția încetinită a thread-urilor individuale, un thread care este gata de execuție fără stall-uri va fi întârziat de instrucțiunile de la alte thread-uri.

Granularitate grosieră – o versiune de hardware multithreading care permite comutarea între thread-uri doar după apariția unor evenimente semnificative – un miss de cache.

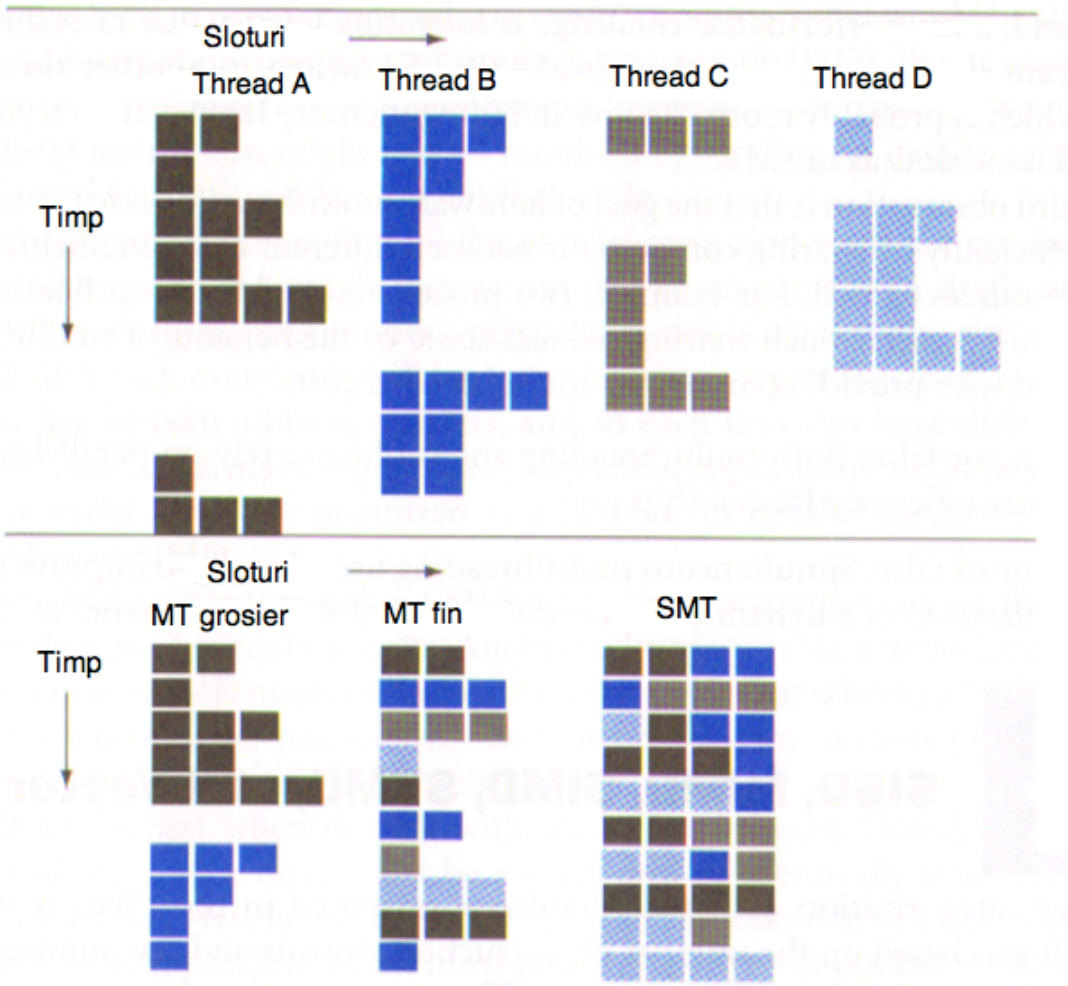
Cel mai mare dezvantaj ala acestei metode este faptul că nu se mai poate ascunde pierderea de throughput.

Limitarea apare din costul necesar pentru pornirea pipeline-ului. Pipeline-ul va trebui golit sau înghețat. Noul thread care începe execuția după stall, trebuie să umple pipe-ul înaintea terminării instrucțiunilor.

Multithreading simultan – SMT – o versiune de multithreading care micșorează costurile multithreading-ului prin utilizarea de resurse necesare pentru probleme multiple , programarea dinamică a microarhitecturii.

Programarea dinamică a procesorului – exploatează paralelismul la nivel de thread precum și paralelismul la nivel de instrucțiune.

Ideea esențială este că SMT-urile au mai multe unități funcționale paralele disponibile decât poate utiliza un thread.



Intel Nehalem multicore suportă SMT cu 2 thread-uri