# Legea lui Sun-Ni

- Ia in considerare limitarea memoriei per procesor
  - Memory-bound speedup
- Relevanta pentru sisteme HPC de mari dimensiuni unde limitarea este data de bandwidth-ul memoriei
- Formalizarea legii Sun-Ni: $T_p >= T_s / p + T_m$
  - Cu $T_m$ timpul necesar de comunicare intre memoriile procesoarelor
  - $T_s$ / p reprezinta speedup-ul ideal fara constrangeri de memorie
- Speedup = G(M) * p
  - G(M) = functie de utilizare a memoriei per procesor
  - p = numarul de procesoare din sistem





# Control Speedup Memory-bound

- Metode de control
  - Utilizarea structurilor de date distribuite
  - Utilizarea mecanismelor de caching
  - Utilizarea compresiei de date
  - Utilizarea comunicatiilor asincrone
- Toate metodele de control presupun compromisuri
- Memoria este o resursa critica in cadrul sistemelor de calcul paralel
  - Cerintele de memorie sunt importante pentru designul algoritmilor paraleli eficienti

## Limitari de memorie prin Sun-Ni

- Baze de date
  - Performanta accesarii BD limitata de memoria sistemului
  - Daca BD nu intra in memorie, citirea de pe disk va scadea timpul total de acces
- ML
  - Multiple modele ML necesita multa memorie (SVM, RF)
  - Antrenarea in batchuri reduce viteza de antrenare
- HPSC
  - CFD, MD necesita multa memorie pentru simulari
  - Daca datele nu intra in memorie, aplicatiile trebuie sa faca checkpointing pe disc crescand timpul de acces si reducand performanta simularii

*120*



## Limita lui Worlton

- Reprezinta limita teoretica pentru speedup luand in considerare overheadul de comunicare intre procesoare
- Overheadul depinde de dimensiunea problemei
- Speedup = 1 / (1 + (C * p) / T)
  - C = overhead de comunicare pe unitatea de lucru
  - T = timpul necesar procesarii pe un singur procesor
  - p = numarul de procesoare
- Limita lui Worlton presupune ca overheadul de comunicare e proportional cu numarul de procesoare si dimensiunea datelor ce trebuiesc comunicate

*121*

# Cresterea Vitezei de Prelucrare

- Trade-off-uri in Limita lui Worlton
  - Efectul sincronizarii poate fi redus prin micsorarea timpului de sincronizare sau prin marirea intervalului intre sincronizari
  - Efectul overhead-ului de comunicare poate fi redus prin reducerea timpului de overhead sau prin cresterea granularitatii task-urilor
  - Cresterea granularitatii task-urilor ajuta la reducerea atat a efectului sincronizarii cat si a overhead-ului.
  - Numarul pasilor de calcul poate fi redus prin
    - Cresterea numarului de procesoare
    - Luand un numar de task-uri multiplu de numarul de procesoare

*122*



# Limitari Worlton

- In practica, speedup-ul e mai mic ca limita Worlton
  - Eficienta algoritmilor
  - Overheadul de comunicare al runtime-systemului paralel
- Examplu
  - Un sistem cu 16 proceoare
  - Overhead de comunicare / unitate de lucru de 0.1ms
  - Timp total de executie serial 100ms
  - Limita Worlton este:
  - Speedup <= 1 / (1 + (0.1 ms * 16) / 100 ms) = 5.88

*123*

# Comparatii Amdahl / Gustafson / Sun & Ni / Worlton



| Lege | Scop | Presupuneri | Limitari |
|---|---|---|---|
| **Amdahl** | Speedup maxim problema fixa (strong scaling) | Divizarea problemei in zone cunoscute (seriale / paralele) | Ignora overheadul de comunicare |
| **Gustafson** | Speedup Maxim pentru problema scalabila (weak scaling) | Dimensiunea problemei e scalabila | Ignora overheadul de comunicare |
| **Sun-Ni** | Speedup luand in considerare capacitatea memoriei | Divizarea problemei in zona memory-bound & non-memory bound | Presupune overhead de comunicare neglijabil |
| **Worlton** | Speedup luand in considerare overheadul de comunicare | Overheadul de comunicare / unitate de lucru si timpul de executie serial sunt cunoscute | Overhead de comunicare proportional cu p si memoria folosita |

*124*

# Comparatie Numerica Amdahl / Gustafson / Sun-Ni / Worlton



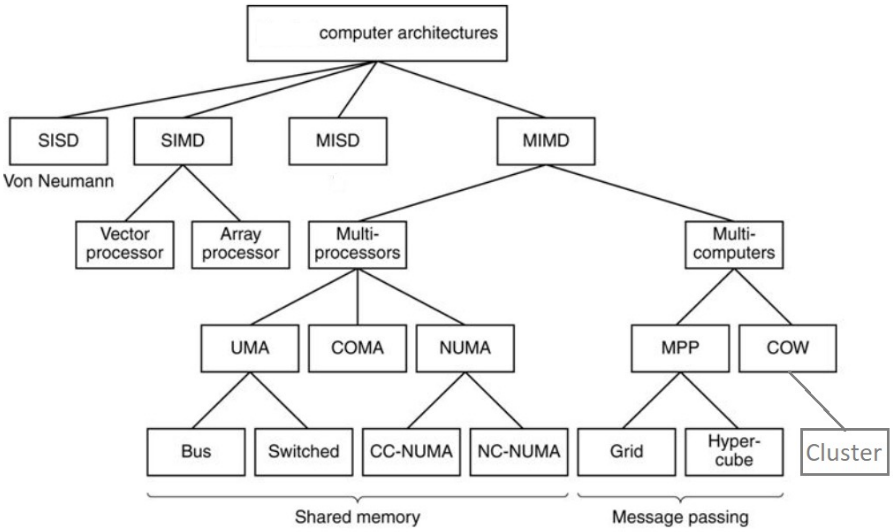
| Lege | *Formula* | *Calcul* | *Limita* |
|---|---|---|---|
| **Amdahl** | S = 1 / (s + (1 - s) / p) | 1 / (0.2 + (1 - 0.2) / 16) | **5** |
| **Gustafson** | S = (1 - s) / s + (1 - s) / p | (1 - 0.2) / 0.2 + (1 - 0.2) / 16 | **6.88** |
| **Sun-Ni** | S = p * G(M) | 16 * G(M) | **16 * G(M)** |
| **Worlton** | S <= 1 / (1 + (c * p) / T) | 1 / (1 + (0.1 ms * 16) / 100 ms) | **5.88** |

Un sistem cu: p = 16 procesoare, sectiune seriala s = 20%, timp executie

T = 100ms si overhead de comunicare / unitate de lucru de c = 0.1ms

*125*

# Arhitecturi Sisteme de Calcul




126



# Classification

| Flynn's Taxonomy (1966-now) | | Nowadays |
|---|---|---|
| **SISD**<br>*Single Instruction*<br>*Single Data* | **SIMD**<br>*Single Instruction*<br>*Multiple Data* | **SPMD**<br>*Single Program*<br>*Multiple Data* |
| **MISD**<br>*Multiple Instructions*<br>*Single Data* | **MIMD**<br>*Multiple Instructions*<br>*Multiple Data* | **MPMD**<br>*Multiple Program*<br>*Multiple Data* |

- Execution models impact the above programming model
- Traditional computer is SISD
- SIMD is *data parallelism* while MISD is pure *task parallelism*
- MIMD is a mixed model (harder to program)
- SPMD and MPMD are less synchronized than SIMD and MIMD
- SPMD is most used model, but MPMD is becoming popular

127

128

## SIMD vs. SPDM

| Caracteristica | SPMD | SIMD |
| --- | --- | --- |
| **Numar de procesoare** | Multiple | Multiple |
| **Flux Intructiuni** | Unic | Unic |
| **Flux Date** | Multiplu | Multiplu |
| **Spatiu de Memorie** | Distribuit | Partajat |
| **Comunicatie** | Comunicare prin Mesaje | Broadcast / Shared |
| **Aplicatii** | HPC, ML, Data Mining | CV, DSP, Vectorial |

*128*

129

## SIMD vs. SPDM

- Ahitecturi SIMD
  - Graphics processing units (GPUs)
  - Vector processing units (VPUs)
  - SIMD units in CPUs
  - Arhitecturi mai eficiente pentru applicatii CV / DSP
- Ahitecturi SPMD
  - Beowulf clusters
  - Distributed shared memory (DSM) systems
  - Message passing interface (MPI) clusters
  - Arhitecturi mai flexibile
- Arhitecturi Hibride
  - Combinatie intre SPMD si SIMD
  - Clustere cu mix de CPU-uri + GPU-uri

*129*

130

# Role of Parallelism?

- Goal of parallel computing
  - **Save time** - reduce wall clock time
    - Speedup - $\frac{\texttt{wall-clock time of serial execution}}{\texttt{wall-clock time of parallel execution}}$
  - **Solve larger problems** - problems that take **more memory than available to 1 CPU**

130

131

# Reduce wall clock time

- Methods
  - Parallelizing serial algorithms (parallel loops)
    - Total number of operations performed changes only slightly
    - Scalability may be poor (Amdahl's law)
  - Develop parallel algorithms
    - Total number of operations may increase, but the running time decreases
- Work Complexity
  - Serialization: parallel algorithm executed sequentially Serializing parallel algorithm may lead to sub-optimal sequential complexity

131

# Parallel Programming Models

- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability





# Parallel Programming Models

Many languages and libraries exist for creating parallel applications.

***Each presents a programming model to its users.***

During this lecture, we'll discuss criteria for evaluating a parallel model and use them to explore various approaches.

| | | |
|---|---|---|
| OpenMP | Charm++ | Linda |
| PThreads | UPC | MapReduce |
| MPI | STAPL | Matlab DCE |
| Cilk | X10 | OpenCL |
| TBB | Fortress | CUDA |
| HPF | Chapel | … |

## Programming Models Evaluation

**What should we consider when evaluating a parallel programming model?**

- Parallel Execution Model
- Productivity
- Performance
- Portability

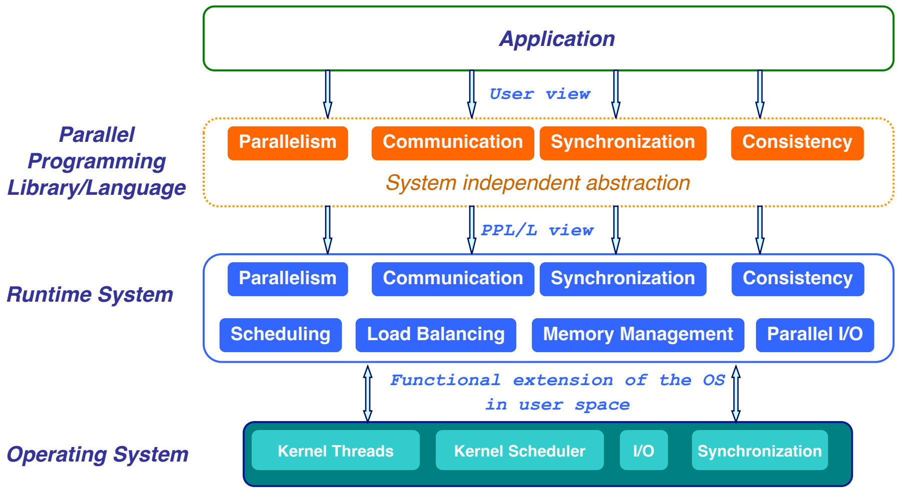




## Parallel Programming Models

- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability

# Parallel Execution Model




136



# Parallel Execution Model

- Parallel Programming Model (user view)
  - Parallelism
  - Communication
  - Synchronization
  - Memory consistency
- Runtime System (RTS)
  - Introduction, definition and objectives
  - Usual services provided by the RTS
  - Portability / Abstraction

137

## Parallel Programming Model (user view)

- Parallelism
- Communication
- Synchronization
- Memory consistency





## PPM – Implicit Parallelism

Implicit parallelism (single-threaded view)

- User not required to be aware of the parallelism
  - User writes programs unaware of concurrency
    - Possible re-use previously implemented sequential algorithms
    - Often minor modifications to parallelize
  - User not required to handle synchronization or communication
    - Dramatic reduction in potential bugs
    - Straightforward debugging (with appropriate tools)
- Productivity closer to sequential programming
- Performance may suffer depending on application
- E.g. Matlab DCE, HPF, OpenMP*, Charm++*

* at various levels of implicitness

# PPM – Explicit Parallelism

Explicit parallelism (multi-threaded view)

- User required to be aware of parallelism
  - User required to write parallel algorithms
    - Complexity designing parallel algorithms
    - Usually impossible to re-use sequential algorithms (except for embarrassingly parallel ones)
  - User responsible for synchronization and/or communication
    - Major source of bugs and faulty behaviors (e.g. deadlocks)
    - Hard to debug
    - Hard to even reproduce bugs
- Considered low-level
  - Productivity usually secondary
  - Best performance when properly used, but huge development cost
  - E.g. MPI, Pthreads

*140*



# PPM – Mixed Parallelism

Mixed view

- Basic usage does not require parallelism awareness
- Optimization possible for advanced users
- Benefits from the two perspectives
  - High productivity for the general case
  - High performance possible by fine-tuning specific areas of the code
- E.g. STAPL, Chapel, Fortress

*141*

# Parallel Programming Models

- Introduction to Programming Models
  - Parallel Execution Models
    - Models for Communication
    - Models for Synchronization
    - Memory Consistency Models
    - Runtime systems
  - Productivity
  - Performance
  - Portability





# PPM – Implicit Communication

Implicit Communication

–Communication through shared variables

–Synchronization is primary concern

– Condition variables, blocking semaphores or monitors

– Full/Empty bit

–Producer/consumer between threads are expressed with synchronizations

–Increases productivity

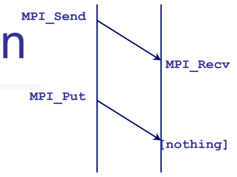
– User does not manage communication

– Reduced risk of introducing bugs

Exec Model | Productivity | Performance | Portability

144

# PPM – Explicit Communication




## Explicit Communication

- Message Passing (two-sided communication, P2P)
  - User explicitly sends/receives messages (e.g., MPI)
  - User required to match every Send operation with a Receive
  - Implicitly synchronizes the two threads
    - Often excessive synchronization (reduces concurrency)
    - Non-blocking operations to alleviate the problem (e.g., MPI_Isend/Recv)
- One-sided communication
  - User uses get/put operations to access memory (e.g., MPI-2, GASNet, Cray T3D)
  - No implicit synchronization (i.e., asynchronous communication)

*144*

Exec Model | Productivity | Performance | Portability

145

# PPM – Explicit Communication

## Explicit Communication – Active Message, RPC, RMI

- Based on Message Passing
- Messages activate a handler function or method on the remote side
- Asynchronous
  - No return value (no `get` functions)
  - Split-phase programming model (e.g. Charm++, GASNet)
    - Caller provides a callback handler to asynchronously process “return” value
- Synchronous
  - Blocking semantic (caller stalls until acknowledgement/return is received)
  - Possibility to use `get` functions
- Mixed (can use both)
  - E.g., ARMI (STAPL)

*145*