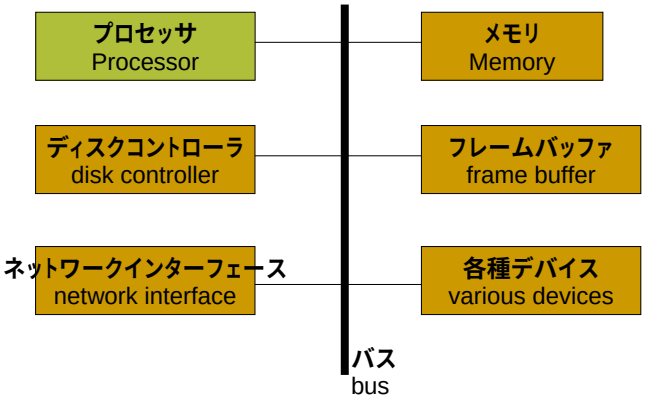
システムプログラム概論
並列コンピュータシステムとそのOS

2007/4/27

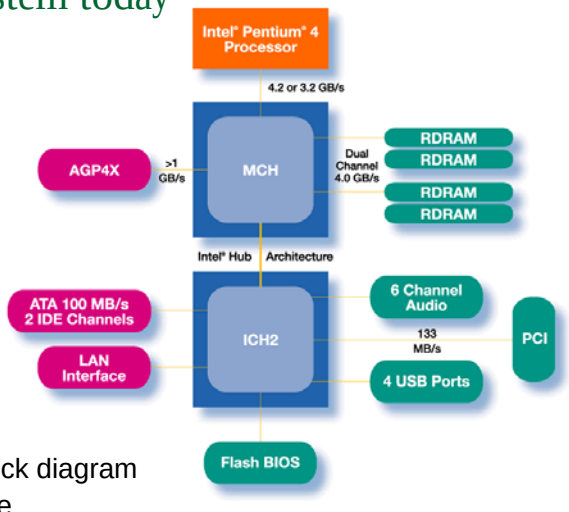
門林　雄基

奈良先端科学技術大学院大学

ユニプロセッサシステム
Uni-processor system revisited

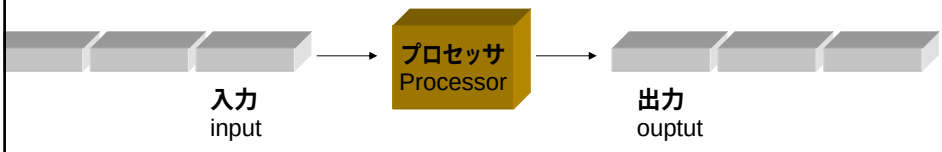


今日のユニプロセッサシステム
Uni-processor system today



- Intel i850 chipset block diagram
- Source: intel web site

ユニプロセッサシステムの処理モデル
Processing model of uni-processor systems

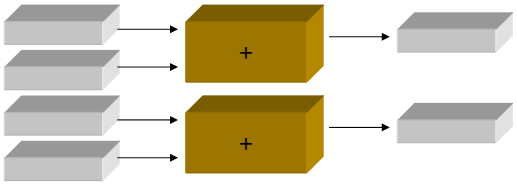


より高速な処理を目指して　－　並列性の導入
Making systems faster: parallelism

- **ビット単位での並列性**
  Bit-level parallelism
  - 8 bit, 16 bit, 32 bit, 64 bit...
- **命令単位での並列性**
  Instruction-level parallelism
  - Pipelining, Superscalar...
    (**本講義の範疇外**)
- **プロセス･スレッド単位での並列性**
  Process-level / Thread-level parallelism

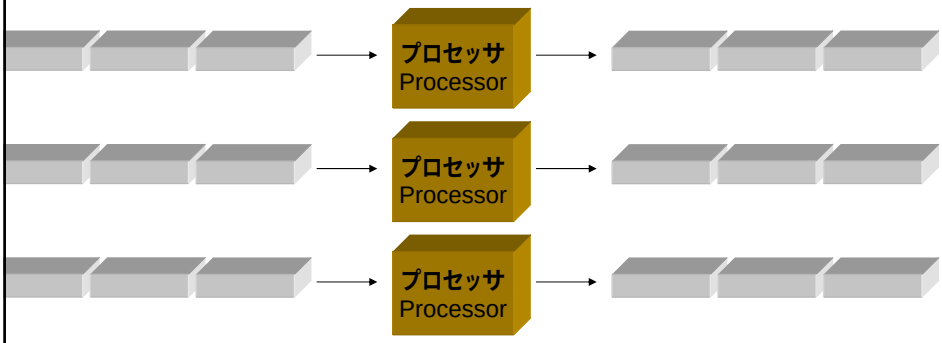
並列システムの処理モデル (1)
Processing model of parallel systems (1)

- SIMD (Single Instruction, Multiple Data)
- SIMD today
  - **プロセッサ内での実現**
  - AltiVec (PowerPC G5), SSE2 (x86)　(example)



並列システムの処理モデル（2）
Processing model of parallel systems (2)

- MIMD (Multiple Instruction, Multiple Data)



MIMD systems: part I

- Centralized memory
  - Symmetric Multiprocessor (SMP)
  - Uniform Memory Access
- Distributed memory
  - Non-Uniform Memory Access (NUMA)
  - today: cc-NUMA

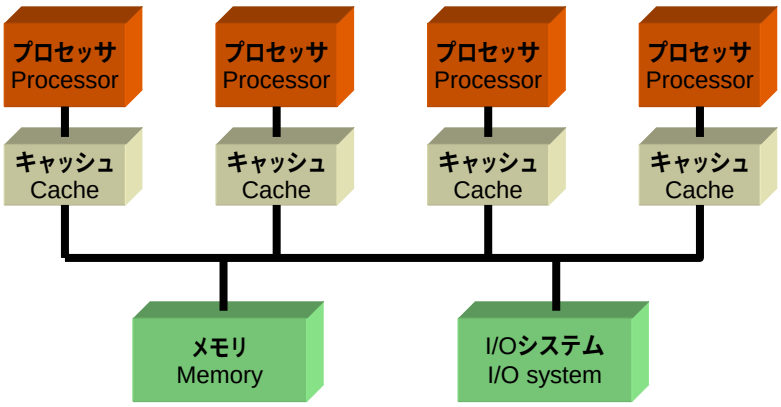
Dual-processor systems

- PowerMAC G5
- Opteron

Dual-core systems

- UltraSPARC IV

Textbook SMP systems



SMP today

- Sun V880
- rx8620

プログラミング・モデル
Programming models

- **マルチプログラミング**
  Multiprogramming
- **共有メモリ**
  Shared memory
- **メッセージ・パッシング**
  Message passing
- **データ・パラレル**
  Data parallel

プログラミング・モデルとOS
Programming models and OS

- multiprocessor operating systems
  - Multiprogramming
  - Shared memory
- uniprocessor operating systems + library
  - Message passing
  - Data parallel

共有メモリ型システムにおけるプログラミングモデル(1)
Programming model in shared memory systems: multiprogramming

プロセスでの並列性 (process-level parallelism)

- **メモリ集中型、分散型いづれにも適用可能**
  applicable to both NUMA and SMP

- C **プログラムから**: fork, exec, waitpid, etc.
- **シェルから**:

```
#!/bin/bash
for ((i = 0; i < 20; ++i)); do
  ./param-search $i
done
```

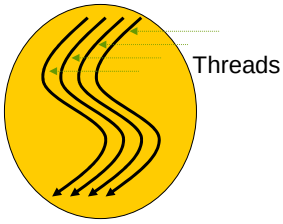
- **パラメータ探索問題** (parameter-search problems) **に向く**
- **タスク並列型の負荷**(task-parallel workloads) **に向く**

共有メモリ型システムにおけるプログラミングモデル(2)
Programming model in shared memory systems: threads

スレッドレベルでの並列性 (thread-level parallelism)

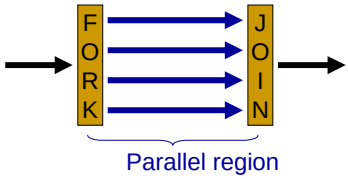
- **共有メモリが必要**
  prerequisite: shared memory
  - SMP, cc-NUMA

- **例**1) posix threads
  - example



スレッドレベルの並列性（例2）: OpenMP
Thread-level parallelism by example: 2/2

- OpenMP
- fork-join model
- parallel region
- Example
- www.openmp.org




```
#include <omp.h>
#define N     1000

main()
{
  int i;
  float a[N], b[N], c[N];
  // ...
#pragma omp parallel shared(a,b,c) private(i)
  {
#pragma omp for schedule(dynamic,100)
    for (i = 0; i < N; i++)
      c[i] = a[i] + b[i];
  }
}
```

共有変数　スレッド固有　分割サイズ　Parallel region

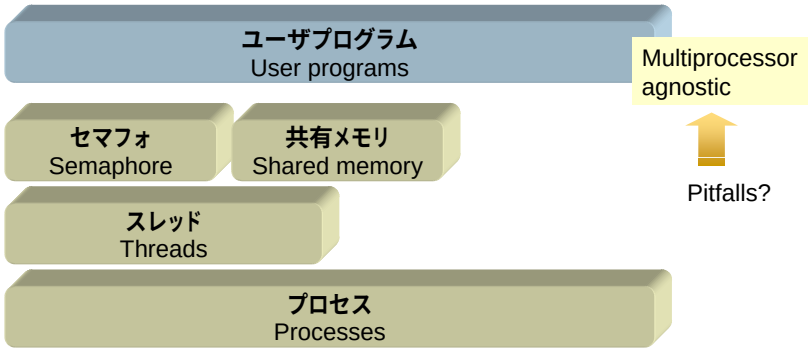
共有メモリ型システムにおけるプログラミングモデル(3)
Programming model in shared memory systems: IPC

- **プロセス間通信**: inter-process communication
- semaphore, shared memory, message passing, ...

- **Cプログラムから**:
  - sem_init, sem_wait, sem_destroy
  - shmget, shmat, shmdt
  - send, recv

マルチプロセッサシステムの抽象化
Abstracting multiprocessor systems

- Multiprogramming, shared memory:
- **同一プログラム、マルチプロセッサ・システムへの可搬性**
  same program; scales up to multiprocessor systems



共有メモリシステムにおけるOSの役割
Operating system role in shared-memory systems

- **プロセッサの割り当て**
  - **スレッドへ / プロセスへ**
- **通信**
  - semaphore, shared memory, message passing
- **ライブラリとコンパイラの役割が増大**
  - Instruction-level parallelism – optimizing compilers
  - Thread-level parallelism – OpenMP

MIMD systems: part II

- Shared memory systems
  - Centralized memory
    - Symmetric Multiprocessor (SMP)
    - Uniform Memory Access
  - Distributed memory
    - Non-Uniform Memory Access (NUMA)
    - today: cc-NUMA

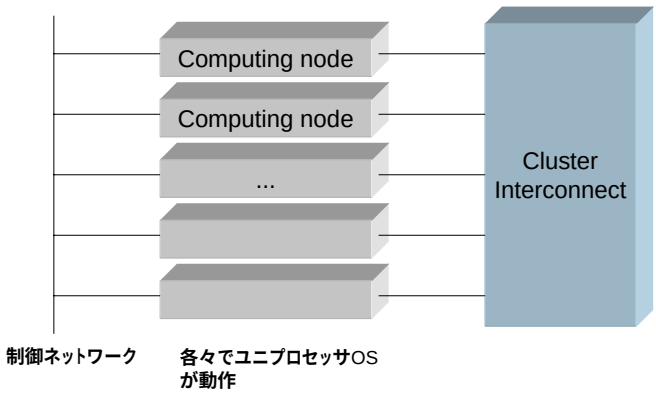
Thread-level parallelism with shared state

Process-level parallelism with shared state

- Multicomputers, Clusters (Shared-nothing*)
  - Research MPPs (dead)
  - Computing cluster
  - Web server cluster, etc.

Process-level parallelism without shared state*

計算機クラスタ
Computing cluster



並列計算機 vs 計算機クラスタ: 隠れたコスト
Hidden cost of computing clusters

- SMP, cc-NUMA
  - **インターコネクトが高価**
  - **メモリアクセス速度の問題**
  - Diminishing returns (IPP p.1)
  - **独自 (商用) OS – 限られた選択肢**

→ Pros
Reproducible results
Reusable software
Bigger memory
Better compiler
Larger MTBF
Reliable support

- Cluster
  - **インターコネクトは安価**
  - **メモリアクセス速度は一定**
  - **高速・安価な市販PCで構成**
  - OS**選択の自由**

→ Cons
Reproducibility issues
Explicit parallelism
Smaller memory
Average compiler
Smaller MTBF
Unreliable support

プログラミング・モデルとOS (再掲)
Programming models and OS revisited

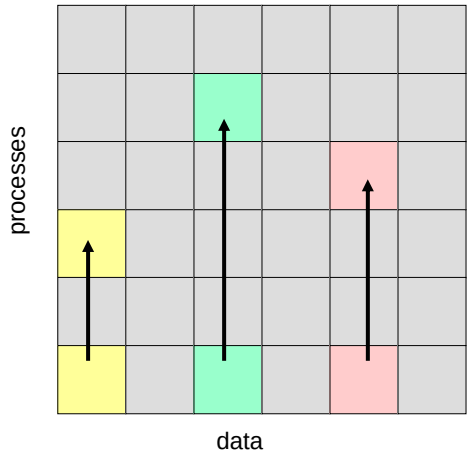
- multiprocessor operating systems
  - Multiprogramming
  - Shared memory
- uniprocessor operating systems + library
  - Message passing
  - Data parallel

Message passing programming model

- MPI (Message Passing Interface)

- Send / receive
- Communication within groups
- Scatter / gather
- Reduce
- Barrier synchronization

Send/Receive in MPI



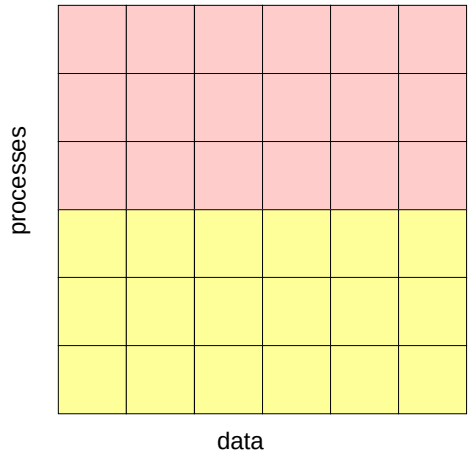
MPI_Send
MPI_Recv

e.g., distributing a very large sparse matrix across nodes

Blocking send/receive, non-blocking send/receive

- Blocking
  - MPI_Send
  - MPI_Recv
- Non-blocking
  - MPI_ISend
  - MPI_IRecv

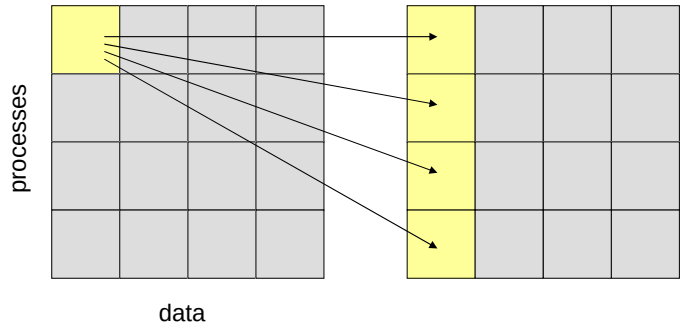
Communication within groups



MPI_Group_*

e.g., running the same algorithm on two data-sets in parallel

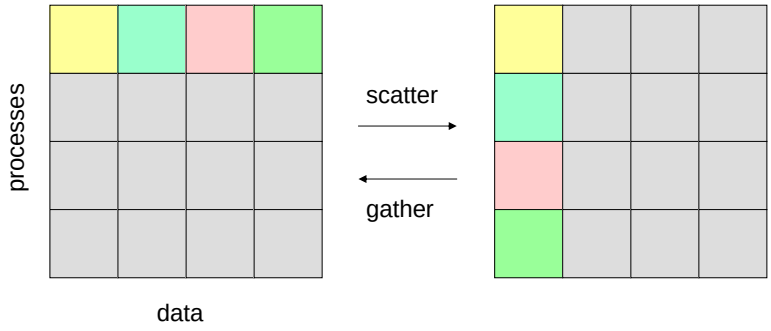
Collective communications(1): Broadcast



MPI_Bcast

e.g., initial parameter

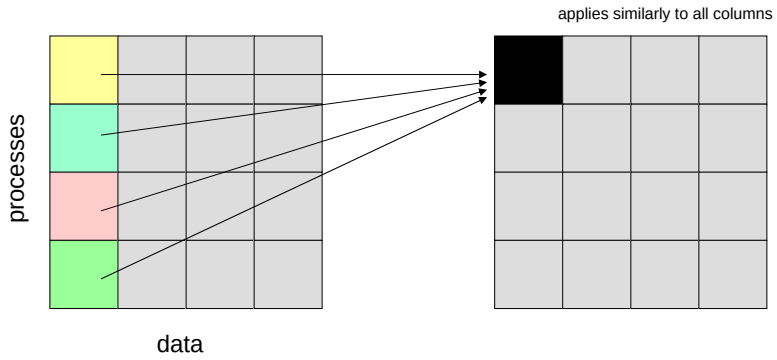
Collective communications(2): Scatter / gather



MPI_Scatter
MPI_Gather

e.g., distribute vector across nodes
matrix multiplication example (uniprocessor, MPI)

Global reduction operations: Reduce



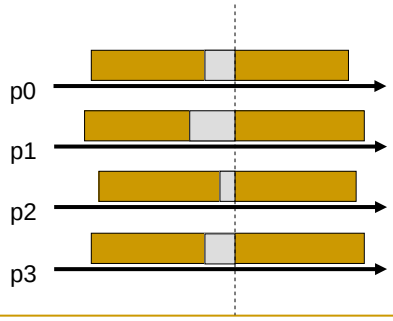
MPI_Reduce

MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN, ...

e.g., vector-matrix product

Barrier synchronization

- MPI_Barrier
- Blocks until all group members have called MPI_Barrier



まとめ

- Taxonomy of:
  - Parallelism
  - Multi-processor systems
- Modern systems:
  - Parallelism everywhere
    - system-level: MIMD
    - insruction-level: SIMD
  - Parallelism-friendly
    - programming models
    - libraries
    - compilers