

Computer Architecture

Computer Science & Engineering

Chương 7

Đa lõi, Đa xử lý & Máy tính cụm





Dẫn nhập

- Mục tiêu: Nhiều máy tính nối lại → hiệu năng cao
 - Đa xử lý (multiprocessors)
 - Dễ mở rộng, sẵn sàng cao, tiết kiệm năng lượng
- Song song ở mức công việc (quá trình)
 - Hiệu xuất đầu ra cao khi các công việc độc lập
- Chương trình xử lý song song có nghĩa
 - Chương trình chạy trên nhiều bộ xử lý
- Xử lý đa lõi (Multicores)
 - Nhiều bộ xử lý trên cùng 1 Chip



Phần cứng & Phần mềm

- Phần cứng
 - Đơn xử lý (serial): e.g., Pentium 4
 - Song song (parallel): e.g., quad-core Xeon e5345
- Phần mềm
 - Tuần tự (sequential): ví dụ Nhân ma trận
 - Đồng thời (concurrent): ví dụ Hệ điều hành (OS)
- Phần mềm tuần tự/đồng thời có thể đều chạy được trên phần đơn/song song
 - Thách thức: sử dụng phần cứng hiệu quả



Lập trình song song

- Phần mềm song song: vấn đề lớn
- Phải tạo ra được sự cải thiện hiệu suất tốt
 - Vì nếu không thì dùng đơn xử lý nhanh, không phức tạp!
- Khó khăn
 - Phân rã vấn đề (Partitioning)
 - Điều phối
 - Phí tổn giao tiếp

Định luật Amdahl

- Phần tuần tự sẽ hạn chế khả năng song song (speedup)
- Ví dụ: 100 Bộ xử lý, tốc độ gia tăng 90?

- $T_{\text{new}} = T_{\text{parallelizable}}/100 + T_{\text{sequential}}$

$$\text{Speedup} = \frac{1}{(1 - F_{\text{parallelizable}}) + F_{\text{parallelizable}}/100} = 90$$

- Solving: $F_{\text{parallelizable}} = 0.999$
- Need sequential part to be 0.1% of original time



Khả năng phát triển (Scaling)

- Bài toán: Tổng của 10 số, và Tổng ma trận $[10 \times 10]$
 - Tăng tốc độ từ 10 đến 100 bộ xử lý
- Đơn xử lý (1 CPU): $\text{Time} = (10 + 100) \times t_{\text{add}}$
- 10 bộ xử lý
 - $\text{Time} = 10 \times t_{\text{add}} + 100/10 \times t_{\text{add}} = 20 \times t_{\text{add}}$
 - $\text{Speedup} = 110/20 = 5.5$ (55% of potential)
- 100 bộ xử lý
 - $\text{Time} = 10 \times t_{\text{add}} + 100/100 \times t_{\text{add}} = 11 \times t_{\text{add}}$
 - $\text{Speedup} = 110/11 = 10$ (10% of potential)
- Với điều kiện tải được phân đều cho các bộ xử lý



Scaling (tt.)

- Kích thước Ma trận: 100×100
- Đơn Xử lý (1 CPU): $\text{Time} = (10 + 10000) \times t_{\text{add}}$
- 10 bộ xử lý
 - $\text{Time} = 10 \times t_{\text{add}} + 10000/10 \times t_{\text{add}} = 1010 \times t_{\text{add}}$
 - $\text{Speedup} = 10010/1010 = 9.9$ (99% of potential)
- 100 bộ xử lý
 - $\text{Time} = 10 \times t_{\text{add}} + 10000/100 \times t_{\text{add}} = 110 \times t_{\text{add}}$
 - $\text{Speedup} = 10010/110 = 91$ (91% of potential)
- Giả sử tải được chia đều cho tất cả CPU

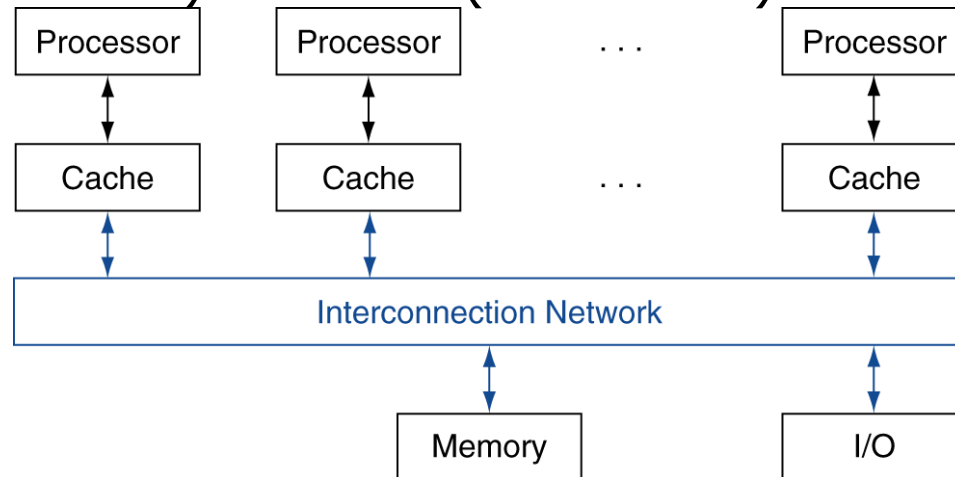


Strong vs Weak Scaling

- Strong scaling: ứng dụng & hệ thống tăng dẫn đến speedup cũng tăng
 - Như trong ví dụ
- Weak scaling: speedup không đổi
 - 10 bộ xử lý, ma trận $[10 \times 10]$
 - Time = $20 \times t_{\text{add}}$
 - 100 bộ xử lý, ma trận $[32 \times 32]$
 - Time = $10 \times t_{\text{add}} + 1000/100 \times t_{\text{add}} = 20 \times t_{\text{add}}$
 - Hiệu suất không đổi

Mô hình chia sẻ bộ nhớ (SMP)

- SMP: shared memory multiprocessor
 - Phần cứng tạo ra không gian địa chỉ chung cho tất cả các bộ xử lý
 - Đồng bộ biến chung dùng khóa (locks)
 - Thời gian truy cập bộ nhớ
 - UMA (uniform) vs. NUMA (nonuniform)





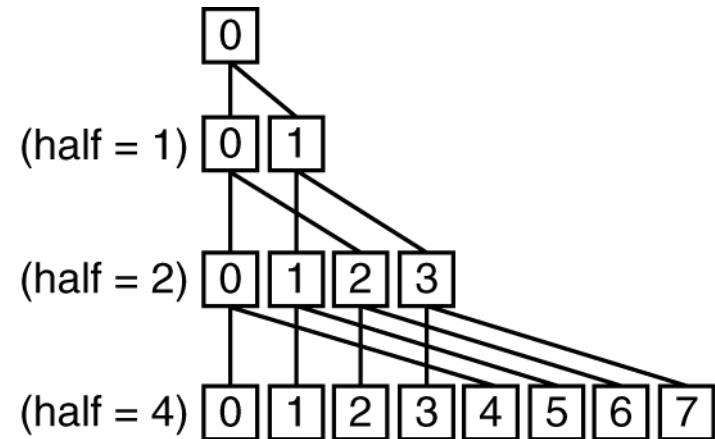
Ví dụ: Cộng dồn (Sum reduction)

- Tính tổng 100,000 số trên 100 bộ xử lý UMA
 - Bộ xử lý đánh chỉ số P_n : $0 \leq P_n \leq 99$
 - Giao 1000 số cho mỗi bộ xử lý để tính
 - Phần code trên mỗi bộ xử lý sẽ là

```
sum[Pn] = 0;
for (i = 1000*Pn;
     i < 1000*(Pn+1); i = i + 1)
    sum[Pn] = sum[Pn] + A[i];
```
- Tính tổng của 100 tổng đơn lẻ trên mỗi CPU
 - Nguyên tắc giải thuật: divide and conquer
 - $\frac{1}{2}$ số CPU cộng từng cặp, $\frac{1}{4}$..., $\frac{1}{8}$..
 - Cần sự đồng bộ tại mỗi bước

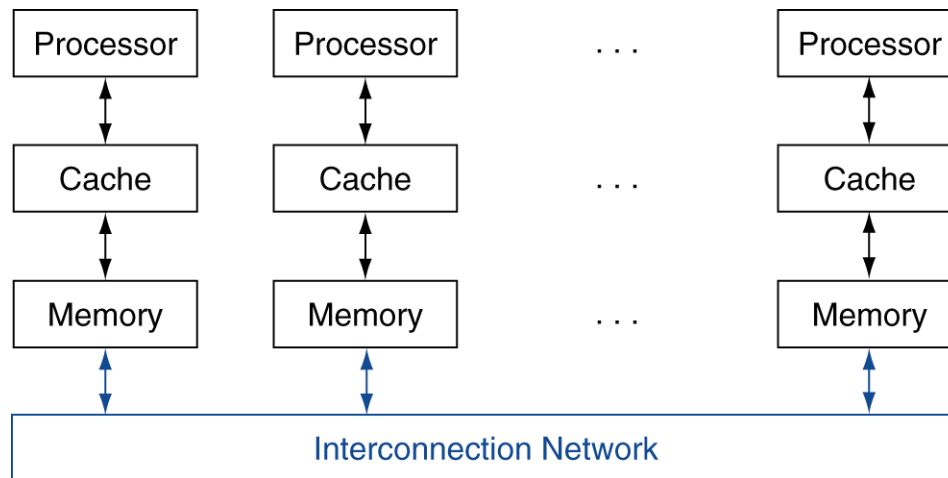
Ví dụ: tt.

```
half = 100;
repeat
    synch();
    if (half%2 != 0 && Pn == 0)
        sum[0] = sum[0] +
        sum[half-1];
        /* Conditional sum needed
        when half is odd;
        Processor0 gets missing
        element */
    half = half/2; /* dividing
    line on who sums */
    if (Pn < half) sum[Pn] =
    sum[Pn] + sum[Pn+half];
until (half == 1);
```



Trao đổi thông điệp

- Mỗi bộ xử lý có không gian địa chỉ riêng
- Phần cứng sẽ gửi/nhận thông điệp giữa các bộ xử lý





Cụm kết nối lỏng lẻo

- Mạng kết nối các máy tính độc lập
 - Mỗi máy có bộ nhớ và Hệ điều hành riêng
 - Kết nối qua hệ thống I/O
 - Ví dụ: Ethernet/switch, Internet
- Phù hợp với những ứng dụng với các công việc độc lập (Web servers, databases, simulations, ...)
- Tính sẵn sàng và mở rộng cao
- Tuy nhiên, vẫn đề nảy sinh
 - Chi phí quản lý (admin cost)
 - Băng thông thấp
 - So với băng thông cử processor/memory trên hệ SMP



Tính tổng

- Tổng của 100,000 số với 100 bộ xử lý
- Trước tiên chia đều số cho mỗi CPU
 - Tổng từng phần trên mỗi CPU sẽ là

```
sum = 0;  
for (i = 0; i < 1000; i = i + 1)  
    sum = sum + AN[i];
```
- Gom tổng
 - 1/2 gửi, 1/2 nhận & cộng
 - 1/4 gửi và 1/4 nhận & Cộng ...

Tính tổng (tt.)

- Giả sử có hàm send() & receive()

```
limit = 100; half = 100; /* 100 processors */
repeat
    half = (half+1)/2; /* send vs. receive
                        dividing line */
    if (Pn >= half && Pn < limit)
        send(Pn - half, sum);
    if (Pn < (limit/2))
        sum = sum + receive();
    limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

- Send/receive cũng cần phải đồng bộ
- Giả sử thời gian send/receive bằng thời gian cộng



Tính toán lưới

- Các máy tính riêng biệt kết nối qua mạng rộng
 - Ví dụ: kết nối qua internet
 - Công việc được phát tán, được tính toán và gom kết quả lại, ví dụ tính thời tiết ...
- Tận dụng thời gian rảnh của các máy PC
 - Ví dụ: SETI@home, World Community Grid



Đa luồng (Multithreading)

- Thực hiện các luồng lệnh đồng thời
 - Sao chép nội dung thanh ghi, PC, etc.
 - Chuyển nhanh ngữ cảnh giữa các luồng
- Đa luồng mức nhỏ (Fine-grain)
 - Chuyển luồng sau mỗi chu kỳ
 - Thực hiện lệnh xen kẽ
 - Nếu luồng đang thực thi bị “khựng”, chuyển sang thực hiện luồng khác
- Đa luồng mức lớn (Coarse-grain)
 - Chuyển luồng khi có “khựng” lâu (v.d L2-cache miss)
 - Đơn giản về phần cứng, nhưng khó tránh rủi ro dữ liệu (eg, data hazards)



Tương lai “đa luồng”

- Tồn tại? Dạng nào?
- Năng lượng tiêu thụ \Rightarrow Kiến trúc đơn giản & Hiệu suất cao
 - Sử dụng các dạng đơn giản đa luồng
- Giảm thiểu thời gian cache-miss
 - Chuyển luồng \rightarrow hiệu quả hơn
- Đa lõi có thể chia sẻ chung tài nguyên hiệu quả hơn (Floating Point Unit or L3 Cache)

Luồng lệnh & Dữ liệu

■ Cách phân loại khác

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345

■ SPMD = Single Program Multiple Data

- Cùng 1 chương trình nhưng trên kiến trúc MIMD
- Cấu trúc điều kiện cho các bộ xử lý thực hiện



SIMD

- Hoạt động trên phần tử vector dữ liệu
 - Ví dụ: MMX and SSE instructions in x86
 - Các thành phần dữ liệu chứa trong các thanh ghi 128 bit
- Tất cả các bộ xử lý thực hiện cùng một lệnh nhưng trên dữ liệu khác nhau
 - Dữ liệu lưu trữ ở các địa chỉ khác nhau.
- Cơ chế đồng bộ đơn giản
- Giảm được phí tổn điều khiển
- Phù hợp với các ứng dụng song song dữ liệu



Bộ xử lý vector

- Cấu tạo từ các bộ phận hoạt động theo cơ chế ống
- Dòng dữ liệu từ/đến các thanh ghi vector vào các bộ phận thực hiện tác vụ
 - Dữ liệu gom từ bộ nhớ vào các thanh ghi
 - Kết quả chứa trong các thanh ghi đưa vào bộ nhớ
- Ví dụ: Mở rộng tập lệnh MIP cho hệ thống vector
 - 32×64 -element registers (64-bit elements)
 - Lệnh Vector tương ứng
 - `lv, sv`: load/store vector
 - `addv.d`: add vectors of double
 - `addvs.d`: add scalar to each element of vector of double
- Giảm đáng kể việc nạp lệnh

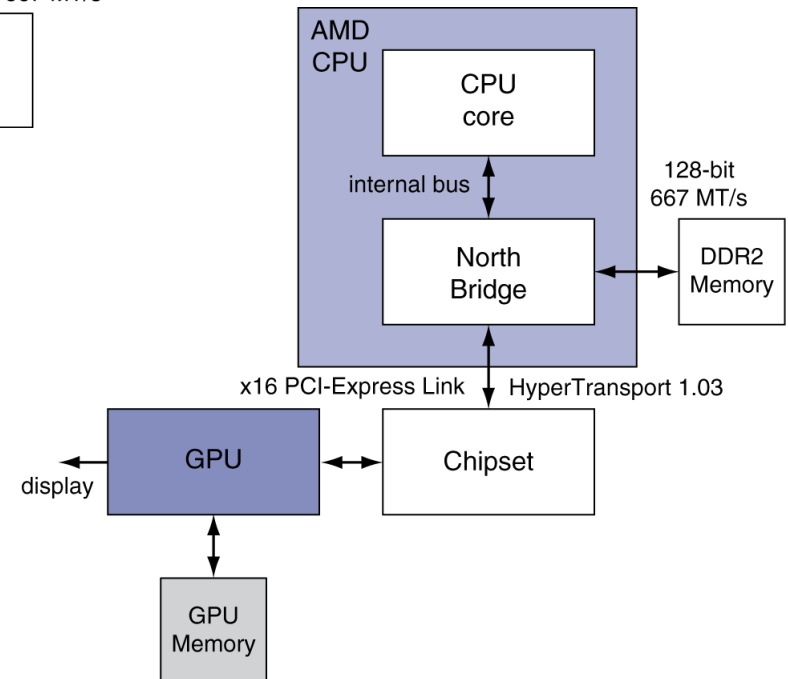
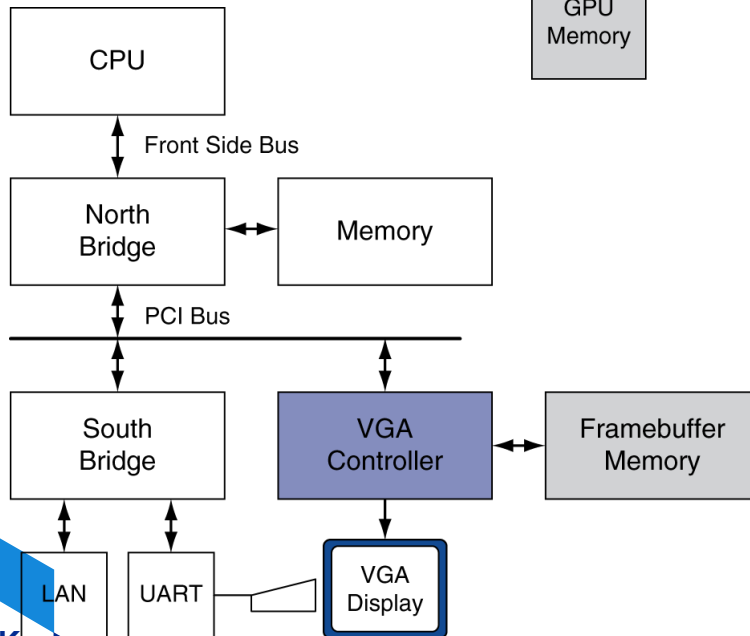
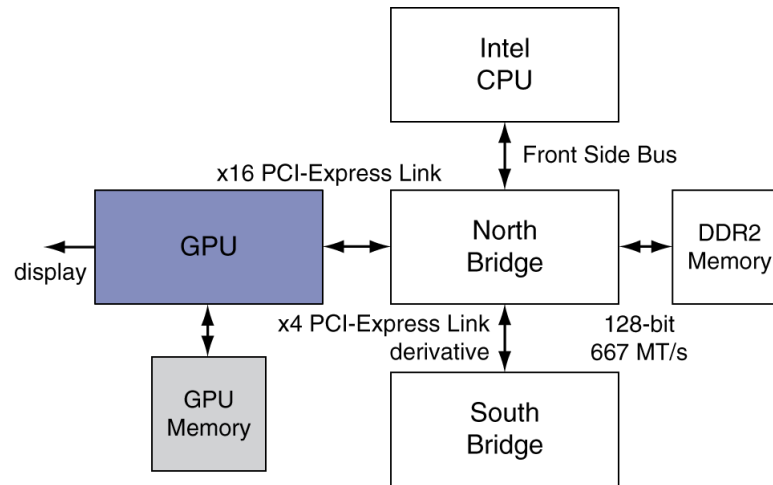


Kiến trúc GPUs

- Trước đây dùng cho video cards
 - Frame buffer memory with address generation for video output
- Xử lý hình 3D
 - Originally high-end computers (e.g., SGI)
 - Moore's Law \Rightarrow lower cost, higher density
 - 3D graphics cards for PCs and game consoles
- Graphics Processing Units
 - Processors oriented to 3D graphics tasks
 - Vertex/pixel processing, shading, texture mapping, rasterization



Đồ họa trong hệ thống





Kiến trúc GPU

- Xử lý ở dạng song song dữ liệu
 - GPUs are highly multithreaded
 - Use thread switching to hide memory latency
 - Less reliance on multi-level caches
 - Graphics memory is wide and high-bandwidth
- Hưởng tới GPU đa năng
 - Heterogeneous CPU/GPU systems
 - CPU for sequential code, GPU for parallel code
- Ngôn ngữ lập trình/APIs
 - DirectX, OpenGL
 - C for Graphics (Cg), High Level Shader Language (HLSL)
 - Compute Unified Device Architecture (CUDA)

Mạng kết nối

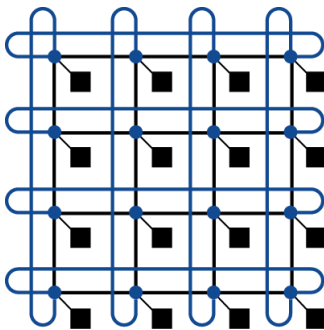
- Cấu hình kết nối mạng (Network topologies)
 - Cấu hình các máy với bộ kết nối và đường truyền



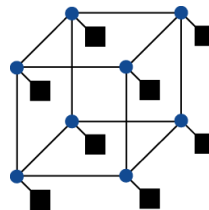
Bus



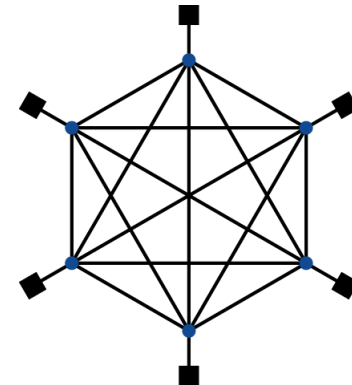
Ring



2D Mesh

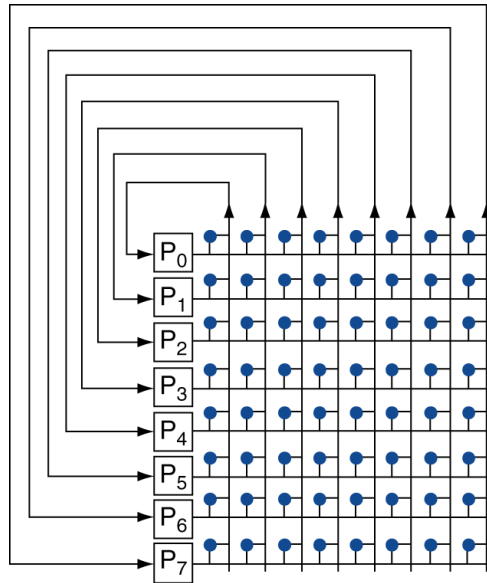


N-cube ($N = 3$)

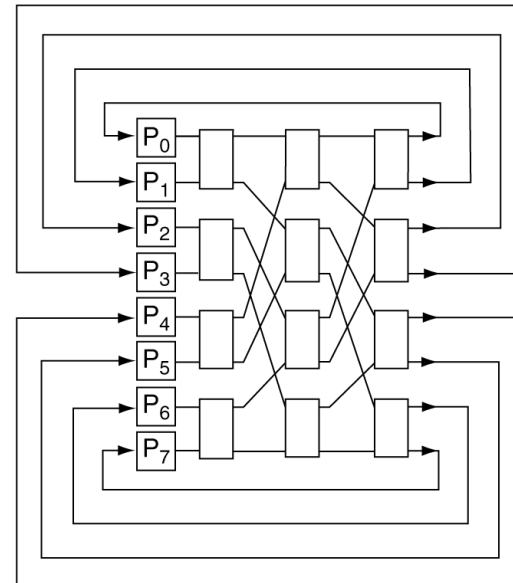


Fully connected

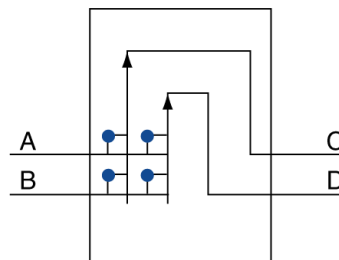
Mạng đa lớp (Multistage)



a. Crossbar



b. Omega network



c. Omega network switch box



Đặc tính mạng

- Hiệu suất
 - Thời gian truyền thông điệp
 - Hiệu xuất đầu ra
 - Bảng thông đường truyền
 - Tổng số bảng thông mạng kết nối
 - Bảng thông 2 chiều
 - Trễ do mật độ đường truyền
- Chi phí
- Nguồn tiêu thụ
- Định tuyến trong mạch

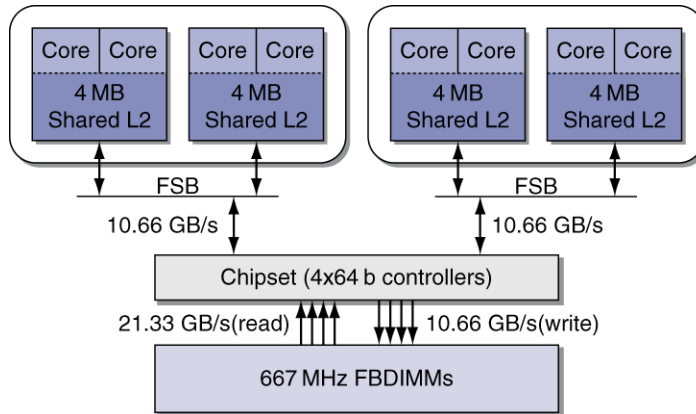


Đánh giá Benchmarks

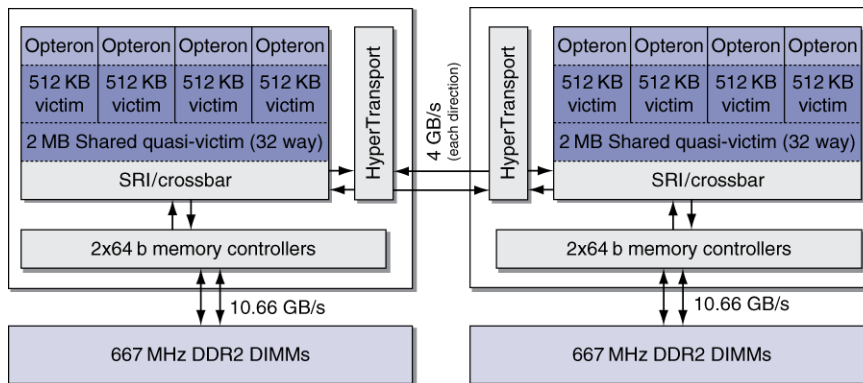
- Linpack: matrix linear algebra
- SPECrate: parallel run of SPEC CPU programs
 - Job-level parallelism
- SPLASH: Stanford Parallel Applications for Shared Memory
 - Mix of kernels and applications, strong scaling
- NAS (NASA Advanced Supercomputing) suite
 - computational fluid dynamics kernels
- PARSEC (Princeton Application Repository for Shared Memory Computers) suite
 - Multithreaded applications using Pthreads and OpenMP



Ví dụ: các hệ thống hiện hành

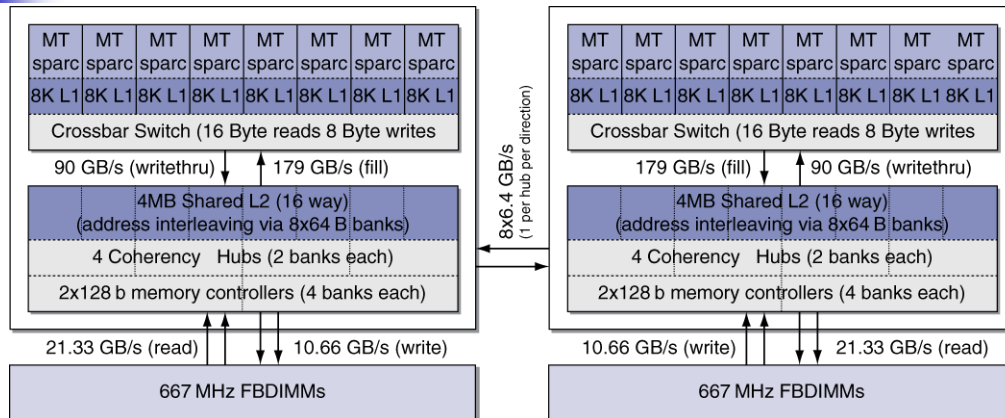


2 × quad-core
Intel Xeon e5345
(Clovertown)

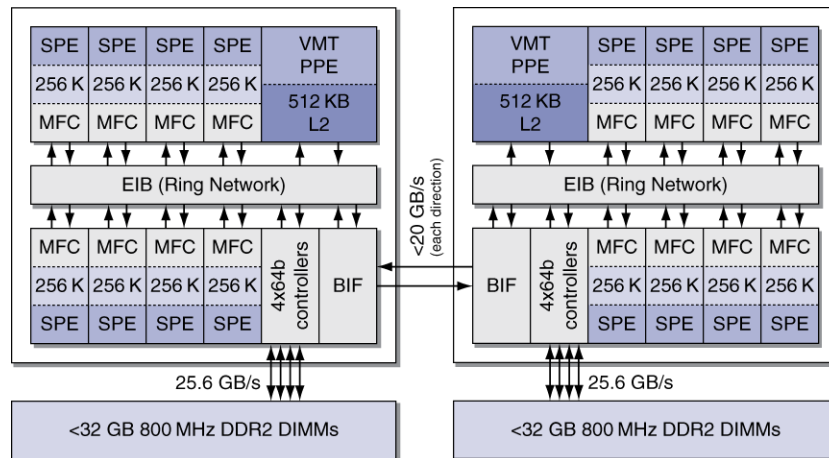


2 × quad-core
AMD Opteron X4 2356
(Barcelona)

Các hệ thống hiện hành (tt.)



2 × oct-core
Sun UltraSPARC
T2 5140 (Niagara 2)



2 × oct-core
IBM Cell QS20



Kết luận

- Mục tiêu: Hiệu suất cao bằng cách sử dụng đa xử lý
- Khó khăn
 - Phát triển phần mềm song song
 - Kiến trúc đa dạng
- Lý do để lạc quan
 - Phát triển phần mềm và môi trường ứng dụng
 - Đa xử lý ở cấp độ chip nhằm giảm thời gian đáp ứng và tăng băng thông kết nối
- Đang còn nhiều thách thức đối với Kiến trúc MT