

병렬형 슈퍼컴퓨터 기본사용법



KISTI 슈퍼컴퓨팅 센터

목 표

KISTI에서 현재 보유하고 있는 병렬형 슈퍼컴퓨터 시스템 및 이의 사용법을 소개함으로써 사용자가 직접 프로그램의 컴파일 및 실행을 할 수 있도록 한다.

차 례

I. 슈퍼컴퓨터와 슈퍼컴퓨팅	4
II. IBM 시스템의 기본 환경	36
III. 컴파일링/디버깅/프로파일링	59
IV. 작업 실행	87
V. 참고 자료	104
VI. 부 록	107

제 I 장

슈퍼컴퓨터와 슈퍼컴퓨팅

- 슈퍼컴퓨터의 개념, 성능의 발전 및 분류에 따른 특징과 구조의 변천 이해
- 고성능 컴퓨팅 및 병렬프로그래밍 모델의 개념 이해

슈퍼컴퓨터의 개념

범용컴퓨터 : 기술적 요인과 경제적 요인에 의해 성능한계 결정

슈퍼컴퓨터 : 기술적 요인만이 성능한계 결정

□ 슈퍼컴퓨터의 필요성

- ❖ 빠른 컴퓨터
 - ❖ 큰 용량
 - ❖ 정확한 해
- 요구 → 당대 기술로 가능한 최고 성능의 컴퓨터 개발

□ 슈퍼컴퓨터의 정의

- ❖ 당대의 기술로 가능한 최고 성능의 컴퓨터
 - 당대의 범용 컴퓨터에 비해 수십 ~ 수천 배 혹은 그 이상의 성능 발휘
 - 기술의 발달로 슈퍼컴퓨터와 범용컴퓨터의 명확한 구분이 모호해짐
 - 현재의 범용컴퓨터가 과거의 슈퍼컴퓨터 성능 발휘 혹은 능가
 - 컴퓨터의 계산성능으로 당대의 순위를 따져서 구분 : TOP500 List

슈퍼컴퓨터의 성능 결정 요인 (1/6)

□ Hz (Machine Cycle)

- ❖ 프로세서의 동작 주기 : 예 - Pentium 1.4GHz
- ❖ 반도체 기술에 의해 좌우되며 이미 기존 기술의 한계에 근접

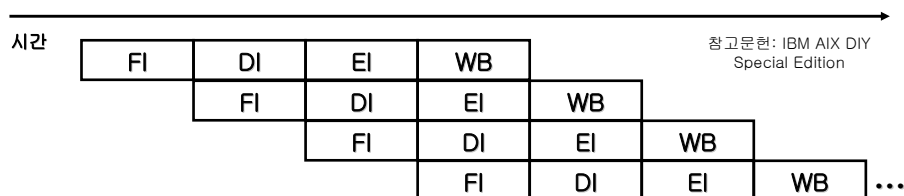
□ CPI (Cycles Per Instruction)

- ❖ 명령어 당 소요 사이클 수
- ❖ 진보된 스칼라 프로세서들의 목표: CPI 값을 낮추는 것
 - CISC : 수십 ~ 수백 CPI 소요
 - 1세대 RISC : 1 CPI 목표
- ❖ 진보된 스칼라 프로세서들의 CPI 값 감소 배경
 - 신기술 적용 : 파이프라이닝, 슈퍼스칼라 기법

슈퍼컴퓨터의 성능 결정 요인 (2/6)

□ 파이프라이닝 기법

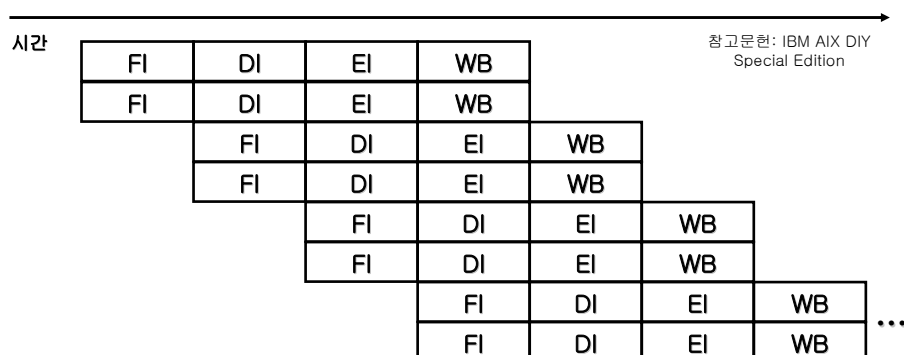
- ❖ 명령어의 병렬처리 기술
- ❖ 명령어 수행 과정 : FI → DI → EI → WB
 - FI (Fetch Instruction) : 명령어 인출
 - DI (Decode Instruction) : 명령어 복호화
 - EI (Execute Instruction) : 명령어 수행
 - WB (Write-Back) : 쓰기



슈퍼컴퓨터의 성능 결정 요인 (3/6)

□ 슈퍼스칼라 기법

- ❖ 파이프라인 기능 유닛을 여러 개 포함
- ❖ 한 사이클 당 여러 개의 명령어가 동시에 처리 됨



슈퍼컴퓨터의 성능 결정 요인 (4/6)

- 관점에 따른 스칼라, 벡터, 슈퍼스칼라의 이해

	스칼라	벡터	슈퍼스칼라
물리적 관점	크기	크기와 방향	---
수학적 관점	$a, 3, \dots$	$A=(a_1, a_2, \dots, a_n)$	여러 스칼라를 동시에 처리
컴퓨터 관점	$a = 1 + 2$ $c = a * b$	$DO\ I = 1, n$ $\quad C(I) = A(I) +$ $\quad B(I)$ $ENDDO \dots$	$f = a*b + c*(d+e)$

슈퍼컴퓨터의 성능 결정 요인 (5/6)

- Flops

- ❖ 초당 수행할 수 있는 부동소수점(floating-point) 연산 회수
- ❖ 단일 프로세서에 대한 이론 성능 Flops 계산:
 - 예 : $5.2 \text{ GFlops} = 1.3 \text{ GHz} \times 4(2 \text{ FPU} \times 2 \text{ (FMA)})$
 - FMA : 각 부동소수점 연산장치(FPU)가 사이클 당 부동소수점 곱셈 연산과 덧셈 연산을 동시에 수행할 수 있는 명령. IBM POWER 계열 프로세서의 특징
 - Hz x 1 (P-III)
 - Hz x 2 (P-IV: SSE2)

- 데이터 전송 능력

- ❖ 계산에 필요한 데이터를 연산장치로 공급하는 능력
- ❖ CPU $\leftrightarrow$ 메모리, 메모리 $\leftrightarrow$ 디스크, 시스템 $\leftrightarrow$ 시스템
 - : 캐시, 인터리빙, 스트라이핑 기법 등을 적용

슈퍼컴퓨터의 성능 결정 요인 (6/6)

❑ 캐시 (Cache)

- ❖ 컴퓨터의 연산 성능을 향상시키기 위해 사용되는 **고속**의 기억 장치
- ❖ 주기억 장치에 비해 프로세서로부터의 **접근 속도**가 월등히 **빠름**
- ❖ 프로그램에서 **빈번히 사용**되는 **데이터**와 **명령어**들을 저장
- ❖ 용량 및 접근 속도가 다른 **Level 1/2/3** 등의 **다단계** 캐시를 사용

❑ 인터리빙 (Interleaving)

- ❖ 디스크 드라이브 혹은 메모리를 좀 더 효율적으로 사용하기 위해 여러 디스크 섹터 혹은 메모리 세그먼트에 **갈치는 데이터**를 접근 시간을 고려해서 **엇갈리게 배치**하는 기법

❑ 스트라이핑 (Striping)

- ❖ 디스크에 저장된 데이터의 빠른 액세스 및 디스크 장애 시 신속한 데이터 가용성을 지원하기 위해 **논리적으로 연속된 데이터**를 **물리적으로 여러 개의 장치**에 나누어 **기록**하는 기법

TOP500 List (1/3)

❑ 가장 빠른 컴퓨터의 세계 500위 순위

- ❖ 1993년 이후 실시
- ❖ **매년** 6월과 11월 Supercomputing Conference를 통해 **2회** 발표 (2004년 11월 24번째 List 발표)
- ❖ <http://www.top500.org>

❑ 성능평가기준: LINPACK 벤치마크

- ❖ N x N 행렬 A로 구성된 연립방정식 $Ax=b$ 의 해를 구하는 성능을 **Flops**(Floating-point operations per second) 단위로 표시
- ❖ HPL : 분산 메모리 시스템을 위한 고성능 LINPACK 벤치마크 패키지
<http://www.netlib.org/benchmark/hpl/>

❑ 슈퍼컴퓨터 시장의 **동향**, 아키텍처 및 관련 기술 발전 방향에 대한 중요한 **지표**

TOP500 List (2/3)

□ TOP500 (2004. 11.)

Rank	Manu- facturer	Computer	Rmax (GFlops)	Installation Site	Country	Year	Processors	Rpeak (GFlops)	Nmax
1	IBM	BlueGene	70720	IBM/DOE	USA	2004	32768	91750	933887
2	SGI	SGI Altix 1.5G, Voltaire Infiniband	51870	Ames Research Center	USA	2004	10160	60960	1290240
3	NEC	Earth-Simulator	35860	Earth Simulator Center	Japan	2002	5120	40960	1075200
4	IBM	eServer BladeCenter JS520	20530	Barcelona Supercomputer Center	Spain	2004	3564	31363	812592
5	California Digital	Intel Itanium2 Tiger4 1.4GHz	19940	Lawrence Livermore National Laboratory	USA	2004	4096	22938	97500
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
500	HP	SuperDome 875MHz Hyperplex	850.6	SBC Service Inc	USA	2004	416	1456	.

자료출처: www.top500.org

□ KISTI IBM 시스템 (병렬형 슈퍼컴퓨터)

❖ 1차 (665.6GFlops) + 2차 (3699.2GFlops) → **4.3TFlops** (이론 성능) 규모

□ KISTI NEC 시스템 (벡터형 슈퍼컴퓨터)

❖ 1차 (80GFlops) + 2차 (160GFlops) → 240GFlops (이론 성능) 규모

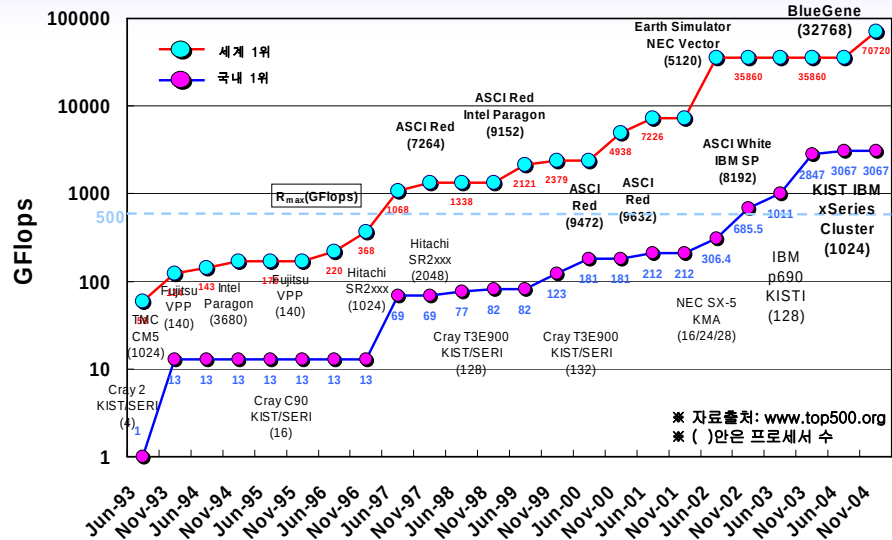
TOP500 List (3/3)

□ TOP500 (2004. 11.) – 국내에 도입된 슈퍼컴퓨터

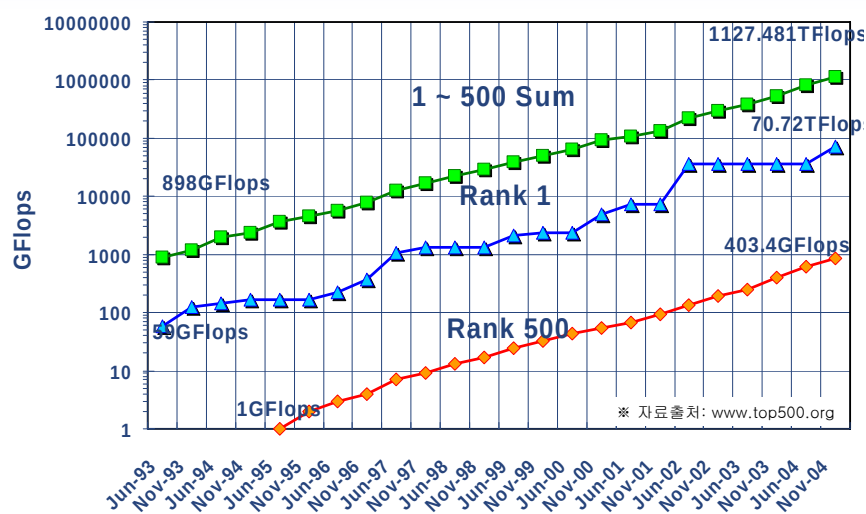
Rank	Manu- facturer	Computer	Rmax (GFlops)	Installation Site	Country	Year	Processors	Rpeak (GFlops)	Nmax
64	IBM	xSeries Xeon 2.4Ghz	3067	KISTI	Korea	2004	1024	4915.2	230000
86	Cray Inc	Cray X1	2188.15	Korea Meteorological Administration	Korea	2004	188	2406.4	.
155	IBM	xSeries Xeon 2.8GHz	1762	KISTI	Korea	2004	512	2867	.
156	IBM	pSeries 690 Turbo 1.7Gh	1760	KISTI	Korea	2003	544	2867	400000
227	Self-made	Pegasus P4 Xeon Cluster 2.2/2.4/2.8 GHz - Giganet - MSWindows	1283	Seoul National University	Korea	2004	520	2557	.
268	HP	Integrity Superdome 1.5Ghz	1210	TelecomSKT	Korea	2004	288	1728	.
310	HP	SuperDome 875MHz / HyperPlex	1138	Zungwon	Korea	2004	576	2016	.
365	HP	SuperDome 875MHz / HyperPlex	1024.3	POSDATA	Korea	2004	512	1792	.
389	HP	SuperDome 875MHz / HyperPlex	1013	Youngwoo	Korea	2004	512	1792	.
390	HP	SuperDome 875MHz / HyperPlex	1013	Zungwon	Korea	2004	512	1792	.
472	HP	SuperDome 875MHz / HyperPlex	1013	Samsung SDS	Korea	2004	448	897	.

자료출처: www.top500.org

슈퍼컴퓨터 성능의 발전 (1/2)



슈퍼컴퓨터 성능의 발전 (2/2)



슈퍼컴퓨터의 구분

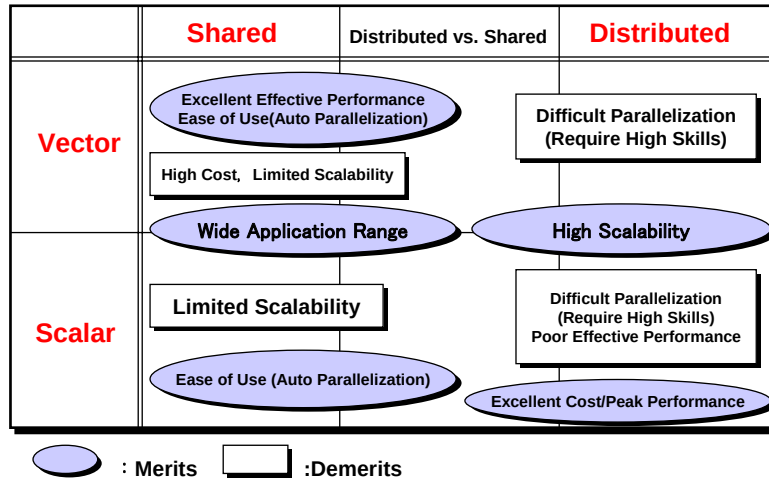
- ❑ 현재의 모든 슈퍼컴퓨터는 다수의 프로세서를 사용
- ❑ 슈퍼컴퓨터를 분류하는 데는 여러 다른 방법이 존재하며, 모든 슈퍼컴퓨터를 명확하게 분류할 수 있는 단일 구분 방법은 없음
- ❑ 프로세서(CPU)에 의한 구분
 - ❖ 벡터 프로세서, 범용 (스칼라) 프로세서
- ❑ 메모리 접근 방식에 따른 구분
 - ❖ 공유 메모리 시스템, 분산 메모리 시스템, 분산 공유 메모리 시스템
- ❑ Flynn에 의한 구분
 - ❖ SISD, SIMD, MISD, MIMD

프로세서에 의한 구분 (1/2)

- ❑ 스칼라 프로세서
 - ❖ 범용 프로세서, 저가
 - ❖ CISC (Complex Instruction Set Computer) : 한 명령어에 메모리 접근, 수치 연산, 기억장치 주소 계산 등 다수의 저 수준 연산을 수행하는 명령어들로 동작하는 프로세서
 - Motorola 680x0, Intel x86 계열
 - ❖ RISC (Reduced Instruction Set Computer) : CISC의 복잡하고 많은 명령어들 대신 빠르게 실행될 수 있는 단순한 명령어들의 조합으로 동작하는 프로세서
 - Berkeley RISC, HP-PA, Clipper, i960, AMD 29000, MIPS R2000, DEC Alpha, PowerPC, IBM POWER 계열
- ❑ 벡터 프로세서
 - ❖ 벡터 데이터의 수치연산 전용 프로세서, 고가
 - ❖ 벡터 프로세싱 : 한 명령어에 벡터 데이터를 동시 처리
 - ❖ NEC Vector, Hitachi SR8xxx, Fujitsu VPP, Cray Vector

프로세서에 의한 구분 (2/2)

- 프로세서 종류 및 메모리 접근 방식에 따른 장단점

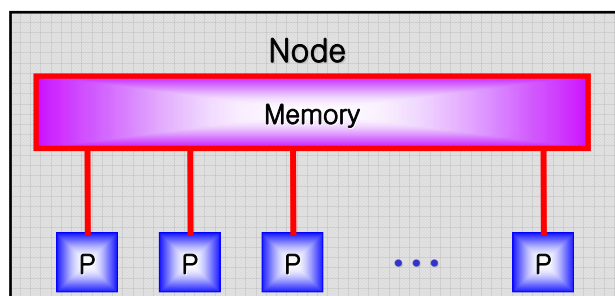


메모리 접근 방식에 따른 구분 (1/4)

- 공유 메모리 시스템

(1) UMA (Uniform Memory Access) : 전형적인 SMP (Symmetric Multiprocessor) 시스템

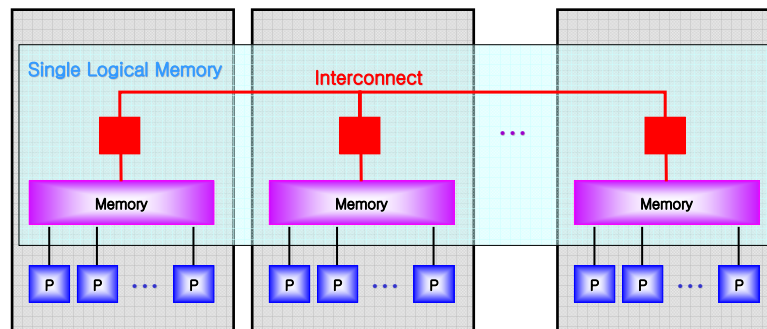
- 논리적, 물리적 공유메모리
- 단일 노드 내 모든 프로세서의 메모리 접근 속도 균일
- 예 : Intel Dual CPU system, Compaq ES40, Sun E10000, HP N-class



메모리 접근 방식에 따른 구분 (2/4)

(2) NUMA (Non Uniform Memory Access) 혹은 CC-NUMA (Cache Coherent NUMA)

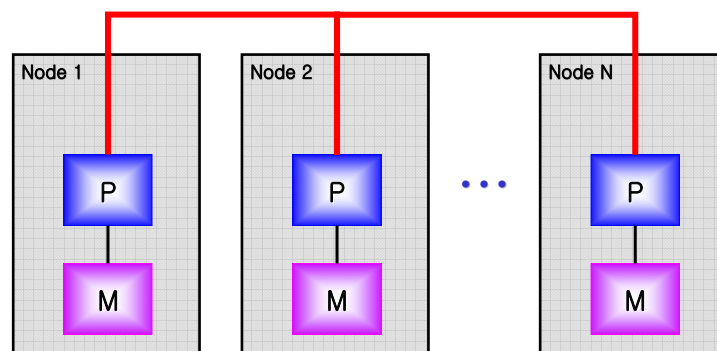
- UMA의 확장성 한계 극복하기 위한 모델
- 논리적 공유메모리, 물리적 분산메모리
- 하드웨어적 연결상태에 따라 메모리 접근 속도 다름 : local vs. remote
- Compaq GS320, HP Superdome, SGI Origin 3000



메모리 접근 방식에 따른 구분 (3/4)

- 분산 메모리 시스템
 - MPP (Massively Parallel Processor)

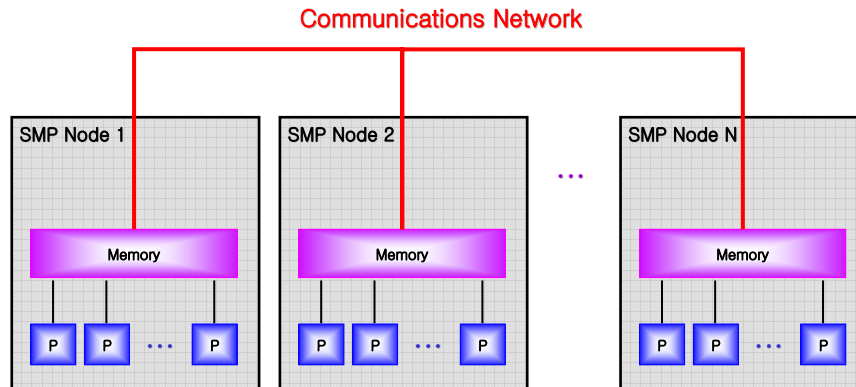
Communications Network



메모리 접근 방식에 따른 구분 (4/4)

❑ 분산 공유 메모리 시스템

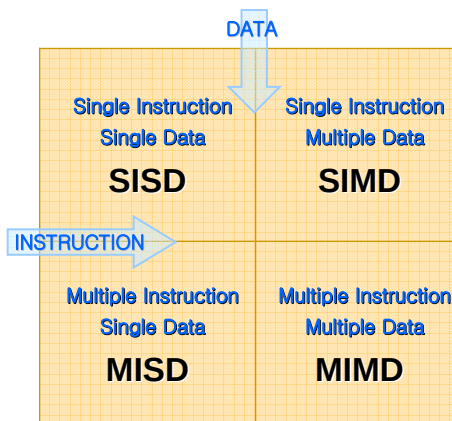
❖ SMP 클러스터



Flynn에 의한 구분

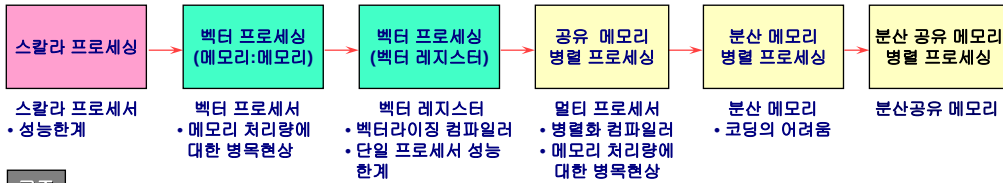
❑ 프로그램의 명령어와 데이터 흐름의 관계에 따라 구분

- ❖ **SISD** : 단일 명령어가 단일 데이터 처리
- ❖ **SIMD** : 단일 명령어가 다수의 데이터 처리
- ❖ **MISD** : 다수의 명령어가 단일 데이터 처리, 실 적용에 없음
- ❖ **MIMD** : 다수의 명령어가 다수의 데이터 처리

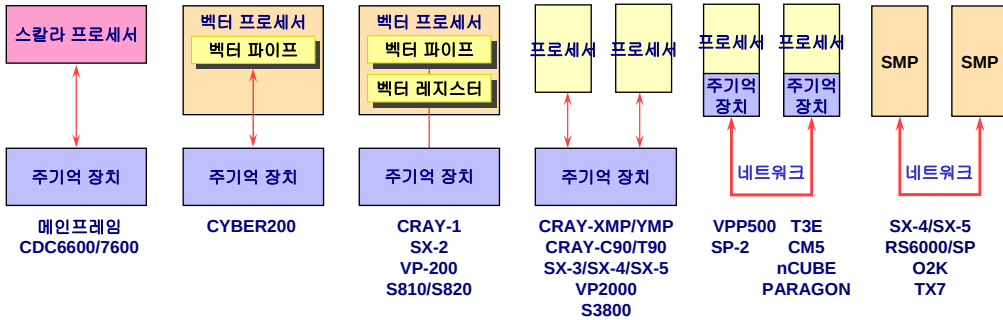


슈퍼컴퓨터 구조의 변천 (1/3)

처리방식



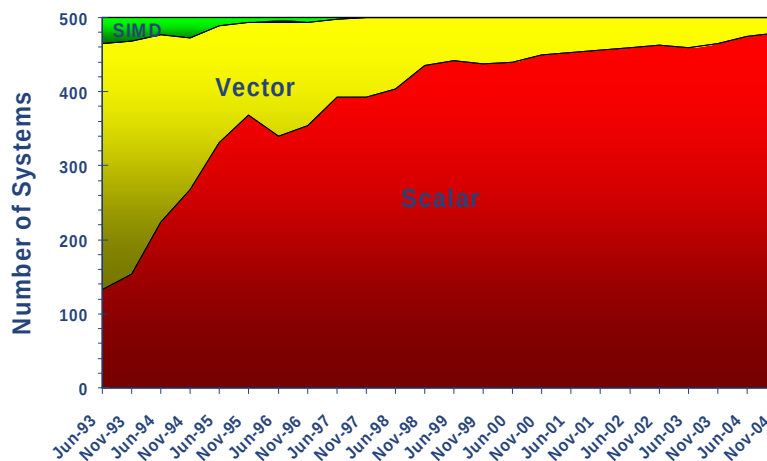
구조



슈퍼컴퓨터 구조의 변천 (2/3)

Processors

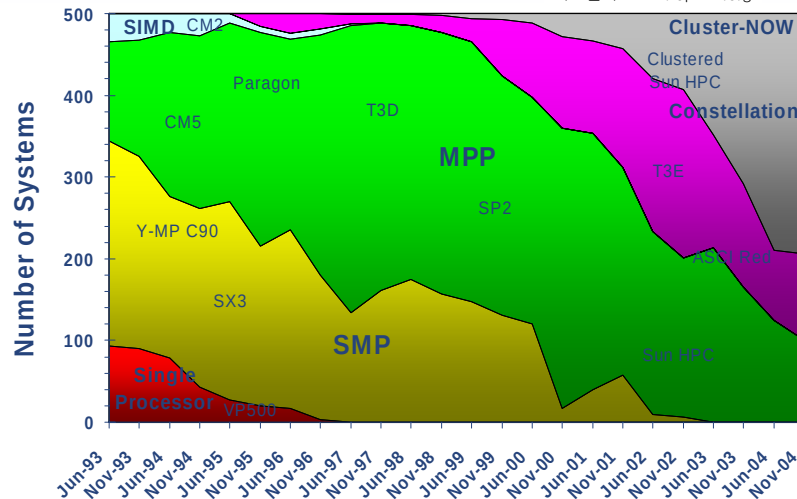
자료출처: www.top500.org



슈퍼컴퓨터 구조의 변천 (3/3)

Architectures

자료출처: www.top500.org



KISTI의 병렬형 슈퍼컴퓨터



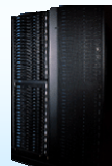
IBM pSeries 690

SMP Cluster, MIMD
CPU 성능 : 1.3, 1.7 GHz
CPU 수 : 672 개
이론 성능 : 4364.8 GFlops
기억 용량 : 4400 GB
메모리 : 분산공유메모리



HP SMP

SMP Cluster, MIMD
CPU 성능: 731, 667, 500 MHz
CPU 수 : 80 개
이론 성능 : 110.9 GFlops
기억 용량 : 80 GB
메모리 : 분산공유메모리



Hamel

SMP Cluster, MIMD
CPU 성능 : 2.8 GHz
CPU 수 : 512 개
이론 성능 : 2.86 TFlops
기억 용량 : 3GB/node
메모리 : 분산메모리

고성능 컴퓨팅 (1/3)

□ 고성능 컴퓨팅의 구성 요소

- ❖ 고성능 하드웨어 : CPU, 메모리, I/O, 네트워크 기술 → 슈퍼컴퓨터
- ❖ 고성능 소프트웨어 : 하드웨어의 성능을 충분히 살릴 수 있도록 설계

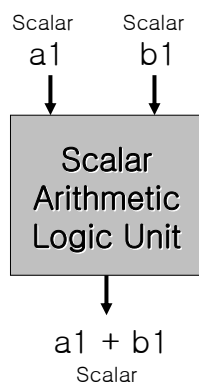
□ 고성능 소프트웨어의 구현

- ❖ 단일 프로세서 성능 최적화
 - 명령어 수준의 병렬성의 극대화 : 파이프라이닝, 슈퍼스칼라, 벡터프로세싱 기법
 - 메모리 액세스 시간의 극소화 : 캐쉬 메모리 사용 최적화
- ❖ 다중 프로세서를 이용한 병렬화
 - 루프/스레드 수준 또는 프로세스 수준의 병렬성
 - 공유/분산 메모리 프로그래밍 모델

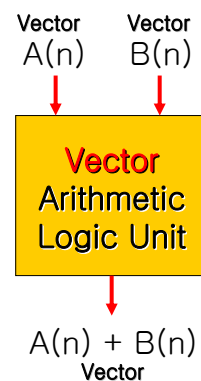
고성능 컴퓨팅 (2/3)

□ 벡터 컴퓨팅

❖ 스칼라 프로세서



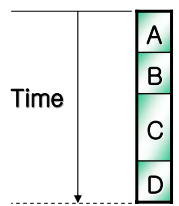
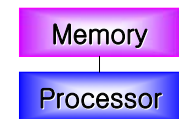
❖ 벡터 프로세서



고성능 컴퓨팅 (3/3)

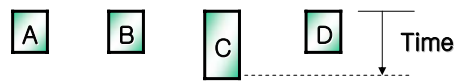
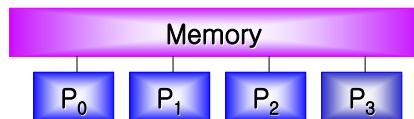
□ 병렬 컴퓨팅

Single Processor



Program

Multiple Processors



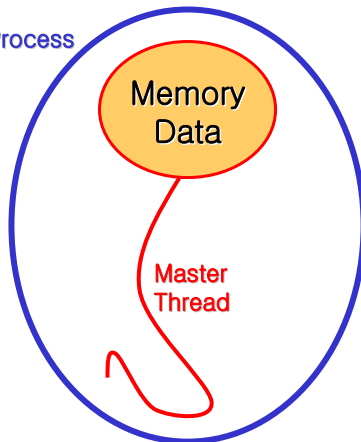
부 프로그램으로 분해
및 병렬 실행

병렬 프로그래밍 모델 (1/4)

□ 스레드와 프로세스

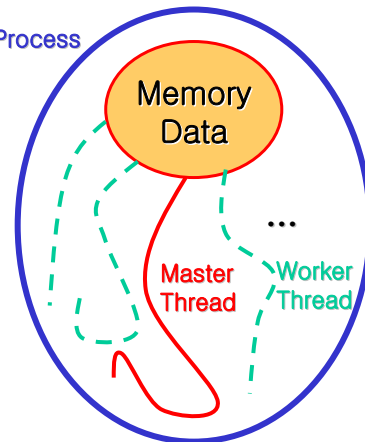
❖ 싱글 스레드 프로세스

Process



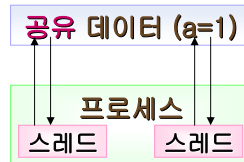
❖ 멀티 스레드 프로세스

Process

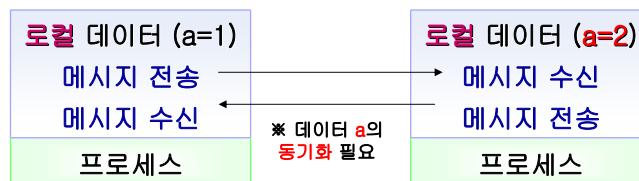


병렬 프로그래밍 모델 (2/4)

- 공유메모리 모델 : OpenMP, POSIX Thread (Pthread) 프로그래밍



- 분산메모리 모델 : MPI, PVM 프로그래밍



병렬 프로그래밍 모델 (3/4)

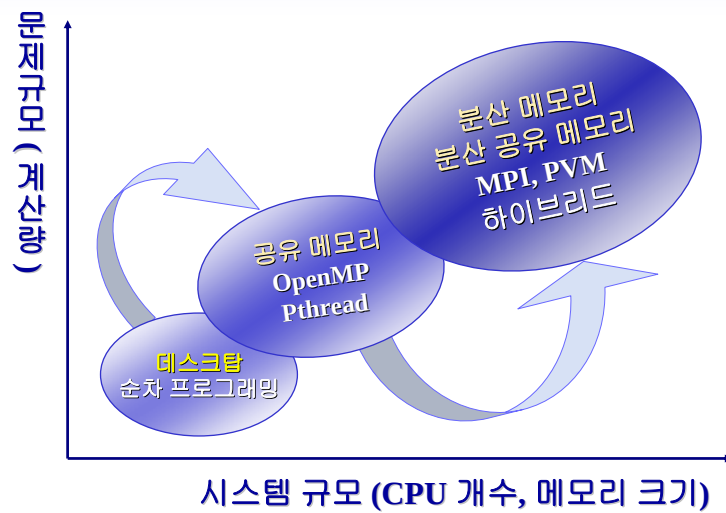
- 메모리 형태별 시스템의 특성 비교

확장성	공유 메모리	<	분산 메모리
가 격	공유 메모리	>	분산 메모리
복잡성	공유 메모리	<	분산 메모리

- 시스템별 적용 모델

구 분	특 징	적용 모델
공유 메모리 시스템 (SMP)	- 공유메모리 병렬 프로그래밍 모델 - 멀티스레드, 루프 수준의 병렬화	- OpenMP (표준) - Pthread (표준)
분산 메모리 시스템 (MPP)	메시지 패싱(Message Passing) 프로그래밍 모델	- MPI (표준) - PVM
분산 공유 메모리 시스템 (SMP 클러스터)	하이브리드 프로그래밍 모델	OpenMP + MPI

병렬 프로그래밍 모델 (4/4)



제 II 장

IBM 시스템의 기본 환경

- KISTI IBM 시스템의 내용/구성 및 기본적인 사용법 이해
- 설치되어 있는 도구 및 프로그램 파악

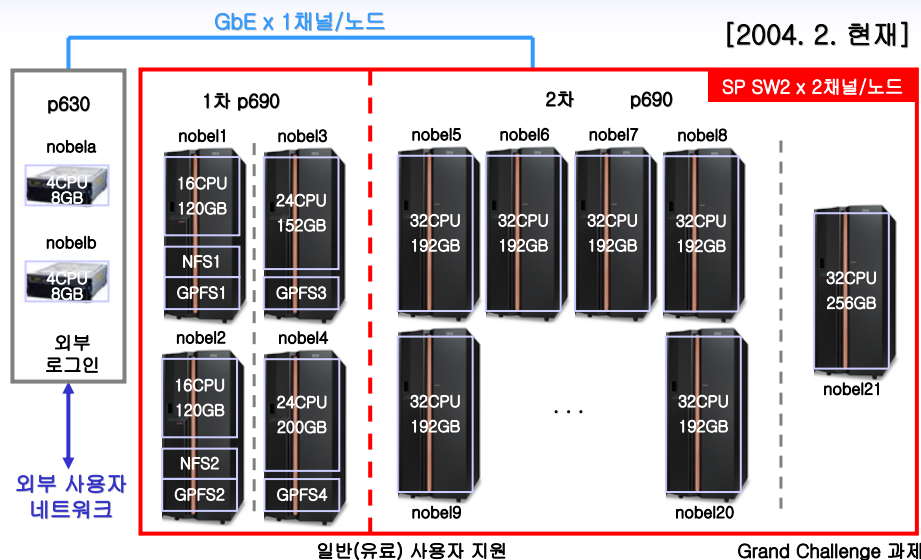
시스템 내용

[2005. 2. 현재]

구 분		컴퓨팅 시스템		로그인 시스템
시스템 명		1차 p690	2차 p690	p630
수 량		4노드	17노드	2노드
CPU	프로세서동작주기 (GHz)	1.3	1.7	1.0
	전체이론최고성능 (GFlops)	665.6 [166.4/노드]	3,699.2 [217.6/노드]	32 [16/노드]
	전체개수 (개)	128 [32/노드]	544 [32/노드]	8 [4/노드]
주기억 장치 용량 (GB)		688 [208×1, 160×3]	3,712 [192×10, 256×7]	16 [8/노드]
외장 디스크 용량 (TB)		98		

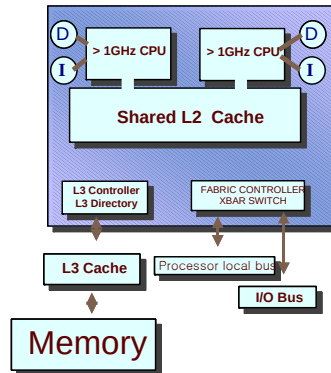
시스템 개략 구성도

[2004. 2. 현재]



POWER4 프로세서 칩의 구조

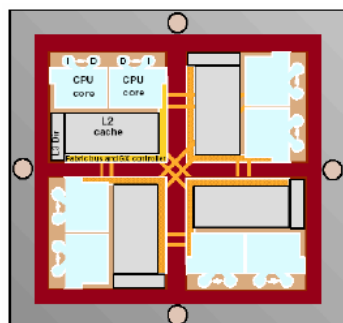
클럭 속도		1.3 / 1.7GHz (p690) 1.0GHz (p630)
단일 칩당 내장 CPU 수		2 개
캐시 구조	L1 캐시	데이터 캐시 용량 : 32KB 명령 캐시 용량 : 64KB
	L2 캐시	용량 : 1.44MB 대역폭 : 100GB/s 이상
	L3 캐시	용량 : 32MB 대역폭 : 10GB/s 이상



POWER4 시스템 기본 구성 단위

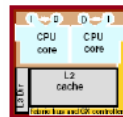
□ MCM (Multi-Chip Module)

- ❖ 4개 칩(8개 CPU)으로 구성
- ❖ IBM p690 : 단위 노드 당 4개 MCM 장착
- ➔ 32 CPU/노드



□ SCM (Single-Chip Module)

- ❖ 1개 칩(2개 CPU)으로 구성
- ❖ IBM p630 : 단위 노드 당 2개 SCM 장착
- ➔ 4 CPU/노드



시스템 접속 (1/3)

□ 노드 구성

노드	IP 주소	용도	비고
nobela.supercomputing.re.kr nobelb.supercomputing.re.kr	150.183.5.101 150.183.5.102	로그인노드 a 로그인노드 b	외부 로그인 (p630)
nobel1.supercomputing.re.kr ~ nobel4.supercomputing.re.kr	-	컴퓨팅노드 1 ~ 컴퓨팅노드 4	일반 사용자 (1차 p690)
nobel5.supercomputing.re.kr ~ nobel20.supercomputing.re.kr	-	컴퓨팅노드 5 ~ 컴퓨팅노드 20	일반 사용자 (2차 p690)
nobel21.supercomputing.re.kr	-	컴퓨팅노드 21	BMT, Grand Challenge (2차 p690)

시스템 접속 (2/3)

□ 일반 접속 : ssh, rlogin

❖ ssh 사용 (예: 사용자 계정 h123abc)

```
$ ssh h123abc@nobelb.ksc.re.kr
```

❖ rlogin, rsh - r* 명령 가능 호스트 시스템 설정

```
$ rlogin nobel1  
$ rsh nobel1
```

시스템 접속 (3/3)

□ 파일 이동 : ftp, sftp, rcp 사용

❖ ftp 사용

```
$ ftp nobela.supercomputing.re.kr
```

❖ sftp 사용 (예: 사용자 계정 h123abc)

```
$ sftp h123abc@nobela.supercomputing.re.kr
```

❖ rcp 사용 : 가능 호스트 시스템 설정

```
$ rcp sample h123abc@nobel1.supercomputing.re.kr:/xtmp1/test/
```

```
$ rcp sample nobel1:/xtmp1/test/
```

* 서로 다른 노드별 스크래치 간에 파일복사가 필요할 때 유용

파일 시스템 (1/5)

시스템 구분	파일 시스템	용량	용도	비고
홈 디렉토리	/edun	1TB	교육계 사용자	사용자별 디스크 사용량 제한 (quota) : 3GB
	/inst	500GB	연구 · 산업계 사용자	
	/system	400GB	내부 사용자	
스크래치 디렉토리	/xtmp1	2TB	노드별 로컬 스크래치 (로그인 노드 140GB)	스크래치 삭제 정책 - 작업종료 후 1일 이상 경과 파일
	/gpfs1 ~ /gpfs4	각 6.4TB	컴퓨팅 노드 GPFS 공유 스크래치	
	/ytmp	400GB	네트워크 공유 스크래치 (nfs1)	

파일 시스템 (2/5)

- ❑ 홈 디렉토리의 사용자별 할당량 및 사용량 파악 명령(ksh shell)

```
$ rsh nfs2 quota
```

Disk quotas for user h123abc (uid 456):

Filesystem	blocks	quota	limit	grace	files	quota	limit	grace
/edun	346592	3000000	3072000		11881	0	0	

- ❖ **blocks** : KB 단위 디스크 용량제한 크기
- ❖ **files** : 최대 허용 파일 개수
- ❖ **quota** : soft limit
- ❖ **limit** : hard limit
- ❖ **grace** : soft limit 초과 hard limit 미만 지속 가능 기간

파일 시스템 (3/5)

- ❑ 홈 디렉토리
 - ❖ NFS (Network File System), 각 노드에서 **Auto Mount** 됨
- ❑ 공유 스크래치 디렉토리
 - ❖ GPFS (General Parallel File System) 공유
 - /gpfs1 ~ /gpfs4, nobel1 ~ nobel21 컴퓨팅노드에서 **공유**
 - I/O 성능이 NFS 방식보다 우수, 병렬 I/O 지원
 - ❖ NFS (Network File System) 공유
 - /yttmp, 모든 노드에서 **Auto Mount** 됨
- ❑ 각 노드별 스크래치 디렉토리
 - ❖ /xtmp1, 각 노드의 로컬 디렉토리
 - ❖ **독립적**으로 운영되며, 순차 I/O 성능이 GPFS 공유 스크래치보다 우수

파일 시스템 (4/5)

- 실습 예제 – Auto Mount 관련
 - ❖ 홈 디렉토리의 Auto Mount 확인

```
$ pwd
[개인 홈 디렉토리]
$ vi sample
## Check for the home directory auto-mounting! ##
$ ls
sample
$ rsh nobel1
$ ls
sample
$ exit
$ rsh nobel2      # nobel3, nobel4, ... , nobel21 각각에 대하여도 확인
```

파일 시스템 (5/5)

- ❖ /ytmp – NFS 공유 스크래치 디렉토리의 Auto Mount 확인
- ❖ /xtmp1 – 로컬 스크래치 디렉토리의 독립적인 운영

```
$ pwd
[개인 홈 디렉토리]
$ ls
sample
$ cp sample /ytmp/[Test_dir]/
$ cp sample /xtmp1/[Test_dir]/
$ ls /ytmp/[Test_dir]/
sample
$ ls /xtmp1/[Test_dir]/
sample
$ rsh nobel1
$ ls /ytmp/[Test_dir]/
sample
$ ls /xtmp1/[Test_dir]/      # /xtmp1은 각 노드별 로컬 디렉토리
ls: xxxx-xxx The file /xtmp1/[Test_dir]/ does not exist.
$ exit
$ rsh nobel2      # nobel3, nobel4, ... , nobel21 각각에 대하여도 확인
```

사용자 작업 환경 (1/2)

- Shell: sh, ksh, csh
 - ❖ POSIX [Bourne] Shell: /bin/sh
 - ❖ Korn Shell: /bin/ksh
/etc/profile → \$HOME/.profile → \$HOME/.env
 - ❖ C Shell: /bin/csh
/etc/csh.cshrc → \$HOME/.cshrc → \$HOME/.login
- 작업 환경 수정
 - ❖ Terminal Type 변경

```
$ export TERM=vt100      # sh, ksh일 경우
% setenv TERM vt100      # csh일 경우
```
 - ❖ 실행 경로 지정

```
$ export PATH=$PATH:/usr/lpp/LoadL/full/bin  # sh, ksh일 경우
% setenv PATH $PATH:/usr/lpp/LoadL/full/bin  # csh일 경우
```
- 환경변수 확인 : echo \$변수명, env 변수명, set

사용자 작업 환경 (2/2)

- 기타
 - ❖ Password 변경

```
$ yppasswd
```
 - ❖ Default shell 변경

```
$ yppasswd -s
```
 - ❖ 명령 라인 에디팅 모드 변경

```
$ set -o vi      # Korn Shell 명령 라인 → vi 에디팅 모드
```
 - ❖ Backspace → Delete 기능 지정

```
$ stty erase [backspace 입력]
```

주요 소프트웨어

구 분	컴퓨팅 노드	로그인 노드
운영 체제	▪ AIX5L V5.1	
병렬 환경 지원	▪ PE V3.2	
배치 작업 관리	▪ LoadLeveler V3.1	
성능 분석	▪ Performance Toolbox V5.1	
프로그래밍 언어	▪ XL Fortran for AIX V8.1 ▪ C for AIX V6.0 ▪ VisualAge C++ Pro for AIX V6.0	
수학 라이브러리	▪ LAPACK, ScaLAPACK ▪ ESSL V3.3, PESSL V2.3 ▪ IMSL(F90 MP), NAG(F77 SMP)	
HPSS	▪ HPSS pftp Client	
상용 소프트웨어	ABAQUS, LS-DYNA3D, CFX, STAR-CD, FLUENT Gaussian, NCAR Graphics	

PE 소개 (1/2)

□ Parallel Environment Software: Version 3.2

- ❖ IBM의 MPI 구현 제품
- ❖ 컴파일러 스크립트, 라이브러리, 디버거, POE
- ❖ MPI-IO 지원: 병렬 파일 I/O
- ❖ MPI 일방 통신 지원
- ❖ MPI 공유 메모리 메시지 전달 지원
- ❖ 32/64 비트 메시지 패싱 지원

PE 소개 (2/2)

□ PE 리스트

메시지 패싱 라이브러리	MPI, MPL
병렬 컴파일 스크립트	mpxlf[90 95], mpcc, mpCC, mpxlf[90 95]_r, mpcc_r
병렬 실행 환경	POE
X-윈도우 병렬 실행 분석도구	Program Marker Array
병렬 파일 복사 유틸리티	mcp, mcpscat, mcpqath
병렬 디버거	pdbx
병렬 프로파일링 도구	xprofiler
병렬 성능 분석 도구	PCT(Performance Collection Tool) UTE(Unified Trace Environment) PVT(Profile Visualization Tool)

공개 라이브러리

□ BLAS (Basic Linear Algebra Subprograms)

- ❖ 기본 행렬 및 벡터 연산 정의
- ❖ Default Library (자동 링크)
 - ➔ 소스 코드가 공개되어 있으나, 각 프로세서 제작사가 고성능 버전 제공함
- ❖ <http://www.netlib.org/blas> 에서 공개된 소스 및 문서 제공
- ❖ PBLAS: BLAS의 분산메모리 버전

□ LAPACK (Linear Algebra PACKage)

- ❖ Fortran77로 작성
- ❖ LINPACK + EISPACK
- ❖ <http://www.netlib.org/lapack> 에서 공개된 소스 및 문서 제공
- ❖ 사용법 : 링크 시 **-llapack** 추가
- ❖ ScaLAPACK : LAPACK의 분산메모리 버전

IBM 시스템 라이브러리

❑ ESSL(Engineering and Scientific Subroutine Library)

- ❖ IBM AIX에 최적화된 과학 계산을 위한 라이브러리
- ❖ LAPACK, BLAS와 호환됨
- ❖ 주요 기능
 - BLAS / Linear Algebraic Equations / Eigensystem Analysis / Fourier Transforms

❑ PESSL(Parallel ESSL)

- ❖ ESSL의 분산메모리 버전
- ❖ ScaLAPACK, PBLAS와 호환됨

❑ 사용 예

```
$ xlf -o sample -O -lessl sample.f
$ mpixlf -o sample -O -lessl -lpeSSL -lblacs sample.f
$ mpixlf_r -o sample -O -lesslmp -lpeSSLmp -lbcassmp sample.f
```

Wrapper의 적용: CCI

❑ CCI (Call Conversion Interface) wrapper

- ❖ **LAPACK** → **ESSL**로 대체 시 사용
- ❖ **IBM 시스템에 최적화**된 성능 기대
- ❖ <http://www.netlib.org/lapack/essl>에서 제공
- ❖ 주의: 링크 시 반드시 **LAPACK**이 먼저 링크되어야 함
- ❖ CCI 적용 예

```
$ xlf -o sample -llapack -lessl sample.f
$ xlf_r -o sample -llapack_r -lesslmp sample.f
```

상용 라이브러리

❑ IMSL(International Mathematical and Statistical Library)

- ❖ <http://www.vni.com/index.html>
- ❖ 수학 및 통계학에 필요한 1,000여 개의 함수로 구성된 상용 라이브러리 (F90 MP)
- ❖ 사용법 : 셸스크립트로 컴파일러, 라이브러리 등 변수 설정 후 사용
\$. /applic/vni/CTT4.0/ctt/bin/cttsetup.sh

❑ NAG (Numerical Algorithms Group)

- ❖ <http://www.nag.co.uk>
- ❖ 수학 및 통계학에 필요한 1,000여 개의 함수로 구성된 상용 Fortran 라이브러리
- ❖ 사용법 : 링크 시 -lnagsmp 추가

HPSS

❑ HPSS(High Performance Storage System)의 개요

- ❖ **대용량 데이터 저장관리 솔루션**
- ❖ 대규모 스토리지 환경을 위한 소프트웨어
 - ➔ 계층적 스토리지 관리 및 서비스 제공
- ❖ KISTI HPSS: 디스크 캐시(3.6TB), 테이프 스토리지(60TB) 제공

❑ 사용법

- ❖ <STEP 1> HPSS 계정 신청 : ykchoi@kisti.re.kr
- ❖ <STEP 2> HPSS 로그인 : pftp_client core 4021
- ❖ <STEP 3> 병렬 파일 store : pput
- ❖ <STEP 4> 병렬 파일 retrieval : pget
- ❖ 보다 자세한 사용법은 부록 참조, 데이터는 tar를 사용하여 묶어서 관리

제 III 장

컴파일링/디버깅/프로파일링

- 순차 프로그램 및 병렬프로그램의
컴파일링/디버깅/프로파일링 방법
습득

개 요

- 컴파일링
 - ❖ 고급 언어 프로그램 → 기계어 프로그램
 - ❖ 컴파일링 + 링킹 → 실행프로그램을 생성
- 디버깅
 - ❖ 프로그램 내의 논리적 오류 발견 및 수정 작업
 - ❖ 컴파일 시 디버깅 옵션 추가 → 디버거 이용
- 프로파일링
 - ❖ 프로그램 실행 중 소요되는 CPU 시간을 서브루틴별로 검토
 - ❖ 실행 시 만들어진 데이터 파일을 프로파일러가 분석
 - ❖ 컴파일 시 프로파일링 옵션 추가 → 프로파일러 이용

컴파일러 소개

언어	Serial	OpenMP	MPI	OpenMP+MPI
C	cc xlc	xlc_r	mpcc	mpcc_r
C++	CC xlC	N/A	mpCC	N/A
Fortran77	f77 xlf	xlf_r	mpxlf	mpxlf_r
Fortran90	xlf90	xlf90_r	mpxlf90	mpxlf90_r
Fortran95	xlf95	xlf95_r	mpxlf95	mpxlf95_r

32비트 / 64비트 모드 프로그램

- 프로그램의 비트 모드 지정
 - ❖ 디폴트는 32비트 모드
 - ❖ OBJECT_MODE 환경변수 32/64
 - ❖ 컴파일러 옵션 -q32/-q64, 아카이버(ar) 옵션 -X32/-X64/-X32_64, 링커지 에디터(ld) 옵션 -b32/-b64
- 32비트 모드 프로그램의 한계
 - ❖ 32비트 주소 표현 : $2^{32} = 4\text{기가}$
 - ❖ 세그먼트 당 256MB(0x10000000)를 다루는 레지스터 16개를 사용
 - ❖ 데이터 영역
 - 디폴트는 1개의 세그먼트를 써서 최대 256MB 사용
 - 최대 8개의 세그먼트를 써서 2GB까지 사용하도록 조정 가능
 - ❖ 256MB 이상의 데이터 → -bmaxdata:BYTES 링크 옵션 사용
 - 예 : -bmaxdata:0x80000000

아키텍처 최적화 컴파일

- ❑ **-qtune=architecture**
 - ❖ 이진코드의 호환성을 유지하면서 지정한 프로세서 아키텍처에 최적화된 오브젝트 코드 생성
- ❑ **-qarch=architecture**
 - ❖ 지정한 프로세서 아키텍처에 의존하는 프로세서 명령어들을 사용해서 최적화 하고자 할때 사용
 - ❖ KISTI 로그인 노드 프로세서 아키텍처는 RS64-IV로 계산노드와 다름
 - 로그인노드에서 디버깅을 목적으로 할 경우 -qarch=pwr4 사용못함
- ❑ **-qcache**
 - ❖ 프로세서 아키텍처의 캐쉬 구조를 지정해서 최적화하는 옵션

순차프로그램 : 컴파일링

- ❑ 컴파일러
 - ❖ f77, xlf, xlf90, xlf95, cc, xlc, CC, xlc
 - ❖ 추천하는 최적화 컴파일 옵션
 - O3 -qarch=pwr4 -qtune=pwr4

- ❑ 컴파일 예

```
$ xlf -o sample -O3 -qarch=pwr4 -qtune=pwr4 sample.f
```

순차프로그램 : 디버깅 (1/2)

□ 디버거

❖ **dbx**: 기본 Unix 행편집기

❖ **xldb**: **GUI** 디버거

➔ <http://www.uni-karlsruhe.de/~SP/manuals/HTML/xldb.html>

❖ 컴파일 시 옵션: **-g**

□ 디버깅 예

\$ dbx [실행파일]	
(dbx) step	# 소스 라인의 함수 안으로 들어가 살펴보는 기능
(dbx) next	# 차례로 살펴보는 기능
(dbx) stop in [procedure]	# breakpoint를 걸어 살펴보는 기능
(dbx) status	# breakpoints를 보는 기능
(dbx) print [expression]	# 값이 제대로 정의되어 있는지 살펴보는 기능
(dbx) rerun	# 재실행
(dbx) quit	# dbx 종료

순차프로그램 : 디버깅 (2/2)

□ 디버깅 예 - 계속

(dbx) continue or c	# breakpoint 혹은 프로그램의 중단시점까지 진행
(dbx) step or s	# 프로시저 콜을 만나면 루틴 안으로 들어가 소스라인 단위로 진행
(dbx) where	# 현재 멈추어 있는 위치 정보 표시
(dbx) stop at	# 소스라인에 breakpoint 지정
(dbx) line 소스라인1,소스라인2	# 소스라인 1부터 2까지의 리스트 출력
(dbx) whatis [variable]	# 변수의 선언 여부 및 선언 상태 표시
(dbx) help	# dbx 명령어 리스트 및 설명

순차프로그램 : 프로파일링 (1/2)

□ prof

- ❖ 프로그램 CPU 사용량 정보 프로파일링 지원
- ❖ 컴파일 시 옵션: **-p**
 - ➔ 컴파일 실행 시 **mon.out** 생성됨

□ gprof

- ❖ 호출된 루틴별 CPU 사용 회수 및 시간
- ❖ 루틴을 호출한 부모 루틴 표시 - 그래프 분석 기능
- ❖ 컴파일 시 옵션: **-pg**
 - ➔ 컴파일 실행 시 **gmon.out** 생성됨

순차프로그램 : 프로파일링 (2/2)

□ xprofiler

- ❖ **gprof의 GUI 버전**
- ❖ 소스 코드 행 수준 프로파일링 가능
- ❖ 컴파일시 **-pg** 옵션 사용
- ❖ 소스 코드 행 수준 프로파일링 시
 - ➔ **-pg** 옵션에 **-g** 옵션을 추가해야 함
- ❖ PE 제품군에 속해 있으며, MPI 병렬 프로그램의 프로파일링 지원

□ 프로파일링 예

\$ prof	# Monitoring Data 파일로 mon.out 이 있어야 함
\$ gprof	# Monitoring Data 파일로 gmon.out 이 있어야 함
\$ xprofiler	# GUI 실행

OpenMP 병렬프로그램 : 컴파일링

❑ 컴파일 옵션

- ❖ **-qsmp=omp**

❑ 멀티스레드 라이브러리 사용

- ❖ **xlf_r, xlf90_r, xlf95_r, xlc_r**

❑ 환경 변수 지정

- ❖ **OMP_SCHEDULE, OMP_NUM_THREADS, OMP_NESTED, OMP_DYNAMIC**

❑ 컴파일 예

```
$ xlf_r -o sample_omp -qsmp=omp sample_omp.f
```

OpenMP 병렬프로그램 : 디버깅/프로파일링

❑ 디버거

- ❖ 순차프로그램의 디버거가 공유메모리 프로그램 지원

- ❖ **dbx, xldb**

- ❖ 컴파일 시 옵션: **-g**

❑ 프로파일러

- ❖ 순차프로그램의 프로파일러가 공유메모리 프로그램 지원

- ❖ **prof, gprof**

- 컴파일 시 각각 **-p, -pg** 옵션 사용

- ❖ **xprofiler**

- 컴파일 시 **-pg** 옵션 사용

- 소스 코드 행 수준 프로파일링 시

- ➔ **-pg** 옵션에 **-g** 옵션을 추가해야 함

MPI 병렬프로그램 : 컴파일

□ 컴파일

- ❖ MPI: mpixlf, mpixlf90, mpixlf95, mpicc, mpCC
- ❖ OpenMP+MPI: mpixlf_r, mpixlf90_r, mpixlf95_r, mpicc_r

□ 컴파일 예

```
$ mpixlf -o para_sample para_sample.f
```

```
$ mpixlf_r -o para_omp_sample -qomp=omp para_omp_sample.f
```

MPI 병렬프로그램 : 디버깅

□ 디버거

- ❖ pdbx
 - 기본 병렬 디버거
- ❖ 컴파일시 -g 옵션 사용

□ 디버깅 예

```
$ pdbx [실행파일]
pdbx(all) step          # 소스 라인의 함수 안으로 들어가 살펴보는 기능
pdbx(all) next          # 차례로 살펴보는 기능
pdbx(all) stop in [procedure] # breakpoint를 걸어 살펴보는 기능
pdbx(all) status        # breakpoints를 보는 기능
pdbx(all) print [expression] # 값이 제대로 정의되어 있는지 살펴보는 기능
pdbx(all) rerun         # 재실행
pdbx(all) quit          # dbx 종료
```

MPI 병렬프로그램 : 프로파일링

□ 프로파일러

❖ xprofiler

- 순차 프로그램 및 병렬 프로그램의 프로파일링 지원
- 컴파일 시 -pg 옵션 사용
- 소스 코드 행 수준 프로파일링 시
- → -pg 옵션에 -g 옵션을 추가해야 함

□ 프로파일링 예

```
$ prof # Monitoring Data 파일로 mon.out이 있어야 함
$ gprof # Monitoring Data 파일로 gmon.out이 있어야 함
$ xprofiler # GUI 실행
```

컴파일 실습 예제 (1/6)

□ 순차 프로그램

- ❖ 예제 디렉토리: /system/center/consult/exam/edul/serial/
- ❖ 예제 파일: hello.f, hello.c

```
$ pwd
[개인 홈디렉토리]
$ mkdir serial
$ cd serial
$ ls /system/center/consult/exam/edul/serial
hello.c hello.f ...
$ cp /system/center/consult/exam/edul/serial/hello.c .
$ cp /system/center/consult/exam/edul/serial/hello.f .
$ ls
hello.c hello.f
```

컴파일 실습 예제 (2/6)

```
$ xlf90 -o hellof hello.f           # FORTRAN 컴파일
** hello === End of Compilation 1 ===
1501-510  Compilation successful for file hello.f.
$ ls
hello.c  hello.f  hellof
$ ./hellof
Hello, World
$ xlc -o helloc hello.c           # C 컴파일
$ ls
hello.c  hello.f  helloc  hellof
$ ./helloc
hello, world
```

컴파일 실습 예제 (3/6)

- OpenMP 병렬프로그램
 - ❖ 예제 디렉토리: /system/center/consult/exam/eduI/omp/
 - ❖ 예제 파일: hellomp.f, hellomp.c

```
$ pwd
[개인 홈디렉토리]
$ mkdir omp
$ cd omp
$ ls /system/center/consult/exam/eduI/omp
hellomp.c      hellomp.f  ...
$ cp /system/center/consult/exam/eduI/omp/hellomp.c .
$ cp /system/center/consult/exam/eduI/omp/hellomp.f .
$ ls
hellomp.c      hellomp.f
```

컴파일 실습 예제 (4/6)

```
$ xlf90_r -o hellompf -qsmp=omp hellomp.f           # FORTRAN 컴파일
** hello  === End of Compilation 1 ===
1501-510  Compilation successful for file hellomp.f.
$ ls
hellomp.c      hellomp.f      hellompf
$ export OMP_NUM_THREADS=4                          #사용 스레드 수 지정
$ ./hellompf
Hello, World from thread = 1
Hello, World from thread = 3
  :      :      :
$ xlc_r -o hellomp -qsmp=omp hellomp.c              # C 컴파일
$ ls
hellomp.c      hellomp.f      hellomp  hellompf
$ ./hellomp
Hello, World from thread = 3
Hello, World from thread = 0
  :      :      :
  :      :      :
```

컴파일 실습 예제 (5/6)

□ MPI 병렬프로그램

- ❖ 예제 디렉토리: /system/center/consult/exam/eduI/mpl/
- ❖ 예제 파일: hellompi.f, hellompi.c

```
$ pwd
[개인 홈디렉토리]
$ mkdir mpi
$ cd mpi
$ ls /system/center/consult/exam/eduI/mpl
hellompi.c      hellompi.f      host.list      ...
$ cp /system/center/consult/exam/eduI/mpl/hellompi.c .
$ cp /system/center/consult/exam/eduI/mpl/hellompi.f .
$ cp /system/center/consult/exam/eduI/mpl/host.list .
$ ls
hellompi.c      hellompi.f      host.list
```

컴파일 실습 예제 (6/6)

```
$ mpixlf90 -o hellompif hellompi.f           # FORTRAN 컴파일
** hello === End of Compilation 1 ===
1501-510 Compilation successful for file hellompi.f.
$ ls
hellompi.c  hellompi.f  hellompif  host.list
$ [poe ]./hellompif -procs 4                 # POE 실행
$ mpicc -o hellompic hellompi.c              # C 컴파일
$ ls
hellompi.c  hellompi.f  hellompic  hellompif  hellompic
$ [poe ]./hellompic -procs 4                 # POE 실행
```

디버깅 실습 예제 (1/7)

□ 순차 프로그램

- ❖ 예제 디렉토리: /system/center/consult/exam/edul/debug/
- ❖ 예제 파일: pi_bug.f

```
$ pwd
[개인 홈디렉토리]
$ mkdir debug
$ cd debug
$ ls /system/center/consult/exam/edul/debug/
pi_bug.f  ...           # bug를 가진 소스 파일
$ cp /system/center/consult/exam/edul/debug/pi_bug.f .
$ ls
pi_bug.f
$ xlf90 -g -o pi_bug pi_bug.f           # -g 옵션 추가 컴파일
$ ls
pi_bug  pi_bug.f
```

디버깅 실습 예제 (2/7)

```
$ ./pi_bug                                     # bug 확인(발견)
Number of steps = 10 and calculated PI = 3.14242598500109871
Number of steps = 100 and calculated PI = -0.118859592570707042E+147
Number of steps = 1000 and calculated PI = -0.118859592570707047E+146
Number of steps = 10000 and calculated PI = -0.118859592570707040E+145
Number of steps = 100000 and calculated PI = -0.118859592570707040E+144
Number of steps = 1000000 and calculated PI = -0.118859592570707038E+143
Number of steps = 10000000 and calculated PI = -0.118859592570707034E+142

$ dbx pi_bug                                    # debugging 시작
Type 'help' for help.
reading symbolic information ...

(dbx) stop in calc_pi                           # breakpoint 설정 [1]
[1] stop in calc_pi

(dbx) stop at 10                                # breakpoint 설정 [2]
[2] stop at "pi_bug.f":10
```

디버깅 실습 예제 (3/7)

```
(dbx) trace sum in calc_pi                      # 변수 trace 설정
[3] trace sum in calc_pi

(dbx) status                                    # breakpoint, trace 확인
[1] stop in calc_pi
[2] stop at "pi_bug.f":10
[3] trace sum in calc_pi

(dbx) cont                                     # breakpoint까지 continue
[1] stopped in calc_pi at line 21 in file "pi_bug.f"
    21  step = 1.0 / num_steps

(dbx) c                                        # continue
initially (at line 22 in "pi_bug.f"): sum = 0.0
after line 24 in "pi_bug.f": sum = 3.9900249376558605
after line 24 in "pi_bug.f": sum = 7.9020053777536603
after line 24 in "pi_bug.f": sum = 11.666711260106602
after line 24 in "pi_bug.f": sum = 15.230185647634443
```

디버깅 실습 예제 (4/7)

```

after line 24 in "pi_bug.f": sum = 18.556588974037769
after line 24 in "pi_bug.f": sum = 21.627606248509938
after line 24 in "pi_bug.f": sum = 24.439557039371099
after line 24 in "pi_bug.f": sum = 26.999557039371098
after line 24 in "pi_bug.f": sum = 29.321763135162101
after line 24 in "pi_bug.f": sum = 31.424259850010984
[2] stopped in main at line 10 in file "pi_bug.f"
    10  print *, 'Number of steps =', num_steps, 'and calculated PI =', pi
(dbx) c
Number of steps = 10 and calculated PI = 3.14242598500109871
[1] stopped in calc_pi at line 21 in file "pi_bug.f"
    21  step = 1.0 / num_steps
(dbx) where                                     # 현 stop 위치 파악
calc_pi(num_steps = 100, pi = 3.1424259850010987), line 21 in "pi_bug.f"
main(), line 9 in "pi_bug.f"

```

디버깅 실습 예제 (5/7)

```

(dbx) status
[1] stop in calc_pi
[2] stop at "pi_bug.f":10
[3] trace sum in calc_pi
(dbx) return                                     # subroutine call return
stopped in main at line 9 in file "pi_bug.f"
    9  call calc_pi(num_steps, pi)
(dbx) c
[2] stopped in main at line 10 in file "pi_bug.f"
    10  print *, 'Number of steps =', num_steps, 'and calculated PI =', pi
(dbx) c
Number of steps = 100 and calculated PI = -0.118859592570707042E+147
[1] stopped in calc_pi at line 21 in file "pi_bug.f"
    21  step = 1.0 / num_steps

```

디버깅 실습 예제 (6/7)

```
(dbx) status
[1] stop in calc_pi
[2] stop at "pi_bug.f":10
[3] trace sum in calc_pi
(dbx) delete 3                                # [3]번 event 삭제
(dbx) status
[1] stop in calc_pi
[2] stop at "pi_bug.f":10
(dbx) whatis sum                               # 변수 definition 확인
real*8 sum
(dbx) print sum                                # 변수 값 출력
-1.1885959257070704e+148
(dbx) assign sum = 0.                          # 변수 값 임시 할당
(dbx) print sum
0.0
```

디버깅 실습 예제 (7/7)

```
(dbx) where
calc_pi(num_steps = 1000, pi = -1.1885959257070704e+146), line 21 in "pi_bug.f"
main(), line 9 in "pi_bug.f"
(dbx) c
[2] stopped in main at line 10 in file "pi_bug.f"
    10  print *, 'Number of steps =', num_steps, 'and calculated PI =', pi
(dbx) n                                       # next (문장 실행 진행)
Number of steps = 1000 and calculated PI = 3.14159273692312269
stopped in main at line 11 in file "pi_bug.f"
    11  enddo
(dbx) quit                                  # dbx 종료
$
$ xlf90 -qinitauto -o pi_nobug pi_bug.f      # automatic variable init.
$ ./pi_nobug
...
```

프로파일링 실습 예제 (1/2)

□ 순차 프로그램

- ❖ 예제 디렉토리: /system/center/consult/exam/eduI/profile/
- ❖ 예제 파일: md_small.f

```
$ pwd
[개인 홈디렉토리]
$ mkdir profile
$ cd profile
$ ls /system/center/consult/exam/eduI/profile
md_small.f      ...
$ cp /system/center/consult/exam/eduI/profile/* .
$ xlf90 -p -o md_small md_small.f           # prof '-p' 옵션
...
$ ./md_small                                # 실행시간 10~20초
4934.71832910619651 ....
```

프로파일링 실습 예제 (2/2)

```
$ ls                                     # mon.out 생성 확인
md_small md_small.f mon.out
$ prof                                   # prof 프로파일링
Name      %Time      Seconds      Cumsecs      #Calls      msec/call
._sin      29.9          4.02         4.02 19839600      0.0002
._cos      21.6          2.90         6.92 14879700      0.0002
.compute   21.3          2.87         9.79 501           5.73
.__mcount  12.8          1.72        11.51
.dist      10.6          1.43        12.94 4959900       0.0003
._sqrt     3.6           0.48        13.42
.cvtloop   0.1           0.01        13.43 1498          0.007
.update    0.1           0.01        13.44 500           0.02
...
$ xlf90 -pg -g -o md_small_pg md_small.f # gprof '-pg' 옵션
...
$ ./md_small_pg                         # 실행시간 10~20초
...
$ ls
md_small md_small_pg md_small.f mon.outgmon.out
$ gprof                                   # gprof 프로파일링 (생략)
```

제 IV 장

작업 실행

- POE 및 LoadLeveler를 사용하는 프로그램 및 작업의 실행방법 습득

프로그램 실행

□ POE를 사용

- ❖ 직접(interactive) 실행
- ❖ MPI 병렬프로그램의 테스트 시 사용을 권장
- ❖ 로그인 노드에서 실행
- ❖ 다수의 동일한 순차프로그램/유닉스 명령어, OpenMP 병렬프로그램을 동시에 실행하고자 할 때도 POE를 이용한 실행이 가능

□ LoadLeveler를 사용

- ❖ 배치작업 실행, 순차 및 병렬프로그램의 실 계산 수행 시에 사용
- ❖ LoadLeveler 명령어들을 사용하여 작업 관리
- ❖ 작업의 대기 및 실행은 작업 큐의 상태에 따라 LoadLeveler가 관리 함
- ❖ Gang 스케줄링 방식으로 설정된 LoadLeveler에서 MPI 프로그램을 실행할 경우 반드시 스레드용 컴파일 스크립트 및 라이브러리 사용
 - mpixlf_r, mpixlf90_r, mpixlf95_r, mpcc_r, mpCC_r

POE 환경 변수 소개 (1/2)

□ POE 환경 변수

- ❖ 형식: MP_변수명
- ❖ POE 명령 옵션으로도 지정 가능
- ❖ 자주 쓰이는 POE 환경 변수 및 명령 옵션
 - MP_PROCS(-procs): 작업에 사용할 태스크 혹은 프로세스 수
 - MP_HOSTFILE(-hostfile, -hfile): 작업에 사용할 호스트들의 리스트가 적혀 있는 호스트리스트 파일 지정
 - MP_SHARED_MEMORY(-shared_memory) : 동일 노드 내에서 실행되는 태스크들 간의 통신에 공유 메모리를 사용할 것인지의 여부 지정, yes 혹은 no
 - MP_EUIDEVICE(-euiddevice) : 태스크들 간의 통신에 사용될 장치(en0, csss) 지정. MP_EUILIB=us인 경우 고속 스위치(csss)로 간주
 - MP_EUILIB(-eulib) : 통신에 사용될 통신 라이브러리 지정, ip 혹은 us. 'us'는 고속 스위치를 사용할 경우만 지정 가능
- ❖ 그 외 부록 참조

POE 환경 변수 소개 (2/2)

□ POE 환경 변수 지정 예

```
$ export MP_SHARED_MEMORY=yes
$ export MP_PROCS=10
$ export MP_HOSTFILE=./host.list
$ poe my_program
```

□ 명령행 옵션 지정 예

```
$ poe my_program -shared_memory yes -procs 10 -hfile ./host.list
```

POE를 사용한 병렬작업 실행

□ SPMD 프로그램 작업

[poe] program_name [POE_option_flags] [executable_arguments]

```
$ poe MPI_PROGRAM -procs N -hostfile HOSTFILE  
$ MPI_PROGRAM -procs N -hostfile HOSTFILE
```

□ MPMD 프로그램 작업 : poe 실행 후 프로그램 순서 명시

```
$ export MP_PGMMODEL=mpmd  
$ export MP_PROCS=4  
$ poe  
0:node1> master  
1:node2> worker  
2:node3> worker  
3:node4> worker
```

POE 실습 예제 (1/2)

□ MPI 프로그램

- ❖ 예제 디렉토리: /system/center/consult/exam/edul/mpi/
- ❖ 예제 파일: hellompi.f, hellompi.c
- ❖ 실행 파일: MPI 컴파일 실습 시 본인 디렉토리 내에 이미 만들었음

```
$ pwd  
[개인 홈디렉토리]  
$ ls  
hellompif    hellompic    ...  
$ cat > host.list # host list file 작성  
nobela  
nobela  
nobela  
nobela  
^D # Ctrl-D (EOF) 입력  
$
```

POE 실습 예제 (2/2)

```
$ ls
hellompif      host.list      ...
$
$ poe ./hellompif -procs 4 -hostfile ./host.list
# MPI 프로그램을 poe 실행 시 'poe' 생략 가능

Hello, World from rank = 0
Hello, World from rank = 2
Hello, World from rank = 1
Hello, World from rank = 3
$
$ poe ./hellompic -procs 3 -hostfile ./host.list
# MPI 프로그램을 poe 실행 시 'poe' 생략 가능

Hello, World from rank = 0
Hello, World from rank = 1
Hello, World from rank = 2
```

LoadLeveler를 사용한 작업 실행 (1/2)

- ❑ 작업관리 절차
 - ❖ 명령행 처리 및 xloadl를 사용하여 GUI 환경에서 처리
- ❑ 작업파일 작성
 - ❖ 보통 쉘 스크립트 형식으로 작성
 - ❖ 작업파일의 LoadLeveler 작업 지시어들은 '#@'으로 시작
- ❑ 자주 사용되는 LoadLeveler 명령어들

\$ llstatus	# 노드 정보 및 상황 출력
\$ llclass	# 작업 클래스 정보 및 상황 출력
\$ llq	# 큐에 있는 작업의 정보 및 상황 출력
\$ llsubmit <i>job_file</i>	# 작업파일을 사용하여 작업 제출
\$ llhold <i>job_id</i>	# 큐에서 대기 중인 작업을 중지 상태로 전환
\$ llhold -r <i>job_id</i>	# 중지 상태인 작업을 대기 상태 전환
\$ llcancel <i>job_id</i>	# 작업의 취소

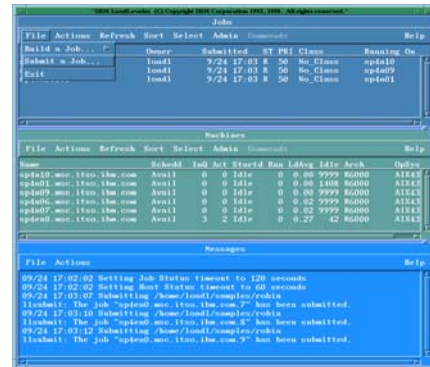
- ❑ 그 외 부록 참조

LoadLeveler를 사용한 작업 실행 (2/2)

□ LoadLeveler X 윈도우 GUI 프로그램 : xloadl

- ❖ DISPLAY 환경변수 설정 : X 클라이언트 측
\$ export DISPLAY=YOUR_SERVER_IP:0.0
- ❖ xhost 연결 허가 : X 서버 측
\$ xhost + nobela.supercomputing.re.kr

xloadl 실행화면



LoadLeveler 클래스 구성 (1/2)

□ 사용 가능 CPU 개수에 따라 분류

- ❖ 순차 작업 용 : 단일 CPU 사용, normal
- ❖ 병렬 작업 용 : 다중 CPU 사용, p_normal_1.3, p_normal_1.7
- ❖ Grand Challenge 과제 / BMT 작업 용 : grand / bmt

※ 2003년 8월 현재 설정

클래스	Max Proc	Wall Clock Limit	총 가용 CPU 수	노드 당 슬롯 수													
				로그인		컴퓨팅											
						일반 사용자								GC		BMT	
				a	b	1	2	3	4	5	6	7	8	9	...	21	
normal	1	480 시간	32	0	0	16	16	0	0	0	0	0	0	0	0	0	0
p_normal_1.3	80	480 시간	80	0	0	16	16	24	24	0	0	0	0	0	0	0	0
p_normal_1.7	96	480 시간	96	0	0	0	0	0	0	32	32	32	0	0	0	0	0
grand	32	960 시간	32	0	0	0	0	0	0	0	0	0	32	0	0	0	0
bmt	416	960 시간	416	0	0	0	0	0	0	0	0	0	0	32	32	32	32
노드 당 총 가용 CPU 수				0	0	16	16	24	24	32	32	32	32	32	32	32	32

LoadLeveler 클래스 구성 (2/2)

- 클래스 별 특징
 - ❖ **normal** : 일반적인 순차 작업 실행
 - ❖ **p_normal_1.3** : 일반적인 병렬 작업 실행, 1차 p690 노드
 - ❖ **p_normal_1.7** : 일반적인 병렬 작업 실행, 2차 p690 노드
 - ❖ **grand / bmt** : Grand challenge과제 / BMT 작업 실행, 2차 p690 노드
- 클래스별 특성치
 - ❖ **Max Proc** : 작업 실행 시 할당받을 수 있는 최대 프로세서 수
 - ❖ **Wall Clock Limit** : 작업이 실행될 수 있는 벽 시계 기준의 제한 시간
 - ❖ **총 슬롯 수** : 최대 실행 가능한 총 프로세스 수
- 사용자 별 제한
 - ❖ 큐 실행 **maxjobs** = 5, 큐 대기 **maxidle** = 2
 - ❖ 큐 대기 + 큐 실행 **maxqueued** = 6

순차프로그램의 LoadLeveler 작업

- 예: /system/center/consult/exam/edul/serial/hello.cmd
- **job_type** = serial 생략 가능

```
#!/bin/ksh
#@ job_type = serial
#@ class = normal
#@ resources = ConsumableCpus(1)
#@ wall_clock_limit = 5:00
#@ queue
./hello > hello.log
```

OpenMP 프로그램의 LoadLeveler 작업

□ 예: /system/center/consult/exam/edul/omp/hellomp.cmd

```
#!/bin/ksh
#@ job_type=serial
#@ class=p_normal_1.7
#@ wall_clock_limit=1:00
#@ notification=never
#@ resources=ConsumableCpus(5)
#@ step_name=hellomp.5
#@ queue
export OMP_NUM_THREADS=5
./hellompf > hellomp.log
```

MPI 프로그램의 LoadLeveler 작업

□ 예: /system/center/consult/exam/edul/mpi/hellompi.cmd

```
#!/bin/ksh
#@ job_type=parallel
#@ notification=complete
#@ class=p_normal_1.3
#@ blocking=unlimited
#@ wall_clock_limit=1:00
#@ Network.MPI=csss,shared,US
#@ total_tasks=5
#@ resources=ConsumableCpus(1)
#@ queue
export MP_SHARED_MEMORY=yes
./hellompif > hellompi.log
```

LoadLeveler 실습 예제 (1/3)

□ Serial/OpenMP/MPI 프로그램

- ❖ 예제 디렉토리: 기본 디렉토리(/system/center/consult/exam/eduI)아래 serial/, omp/, mpi/
- ❖ 예제 파일: job command files '*.cmd'
→ 이 때 실행파일이 각 디렉토리 내에 있어야 함 (컴파일 작업 시 모두 만들어 놓음)
- ❖ 예제 실행: MPI 파일 이용 → Serial, OpenMP 파일도 동일하게 적용됨

```
$ pwd
[개인 홈디렉토리]
$ cp /system/center/consult/exam/eduI/serial/hello.cmd serial
$ cp /system/center/consult/exam/eduI/serial/hellomp.cmd omp
$ cp /system/center/consult/exam/eduI/serial/hellompi.cmd mpi
```

LoadLeveler 실습 예제 (2/3)

```
$ cd mpi
$ ls
hellompi.c      hellompi.f      host.list      hellompif      hellompi.cmd
hellompic
$
$ llstatus
Name                Schedd  InQ  Act  Startd  Run  LdAvg  Idle  Arch  OpSys
nobel1.supercomputing.re Avail  1    1  Busy    16   9.58    9  R6000 AIX51
nobel10.supercomputing.re Avail  0    0  Idle     0   0.07   612  R6000 AIX51
nobel11.supercomputing.re Avail  0    0  Idle     0   0.93    71  R6000 AIX51
:                :
$ llsubmit hellompi.cmd
llsubmit: The job "nobel1.supercomputing.re.kr.436" has been submitted.
```

LoadLeveler 실습 예제 (3/3)

\$ llq

Id	Owner	Submitted	ST	PRI	Class	Running On
nobela.481.0	smlee	3/12 11:36	R	50	p_normal_1.7	nobel7
nobela.574.0	consult	3/16 18:05	ST	50	p_normal_1.3	nobel4

2 job step(s) in queue, 0 waiting, 1 pending, 1 running, 0 held, 0 preempted

\$ llhold nobela.574.0

\$ llhold -r nobela.574.0

\$ llcancel nobela.574.0

llhold 이하는 job이 간단한 관계로 실습이 불가능 함
참고만 하십시오.

제 V 장

참 고 자 료

- 참고 문헌
- 유용한 사이트

참고 문헌

❑ Finding Documentation on the World Wide Web

- http://www.rs6000.ibm.com/resource/aix_resource/sp_books
- <http://www.rs6000.ibm.com/support/sp/resctr/index.html>

❑ Accessing PE Documentation Online

- The HTML files and an index that links to them are installed in the /usr/lpp/ppe.html directory
- Enter "/usr/lpp/ssp/bin/resource_center"
- On unix workstation, you should first export DISPLAY to your own workstation
- On PC, you may need some X-emulator like exceed, xmanager, etc.

❑ SP Software Publications

- IBM Parallel Environment for AIX: Hitchhiker's Guide, SA22-7424-01
- IBM Parallel Environment for AIX: Messages, GA22-7419-01
- IBM Parallel Environment for AIX: MPI Programming Guide, SA22-7422-01
- IBM Parallel Environment for AIX: MPI Subroutine Reference, SA22-7423-01
- IBM Parallel Environment for AIX: Operation and Use, Volume 1, SA22-7425-01
- IBM Parallel Environment for AIX: Operation and Use, Volume 2, SA22-7426-01
- LoadLeveler Diagnosis and Messages Guide, GA22-7882-00
- Using and Administering LoadLeveler, SA22-7881-00

유용한 사이트

❑ Designing and Building Parallel Programs

- <http://www.mcs.anl.gov/dbpp>

❑ OpenMP ARB

- <http://www.openmp.org>

❑ Parallel Programming in OpenMP

- http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-671-8

❑ MPI at Argonne

- <http://www.mcs.anl.gov/mpl>

❑ Batch Compiler and Other Tools (VisualAge C++)

- <http://www-4.ibm.com/software/ad/vacpp/download/pdf/aix/batch.pdf>

❑ XL Fortran Language Reference

- <http://www.ibm.com/software/ad/fortran/xlfortran/pdf/v7r1/lr.pdf>

❑ XL Fortran User's Guide

- <http://www.ibm.com/software/ad/fortran/xlfortran/pdf/v7r1/Ug.pdf>

제 VI 장

부 록

- ❑ HPSS 사용법 예제
- ❑ 주요 소프트웨어 비교표
- ❑ Batch 작업관리 시스템 명령 비교
- ❑ POE 환경 변수
- ❑ LoadLeveler 주요 용어
- ❑ LoadLeveler 작업 명령 키워드

HPSS 사용법 예제 (1/4)

- ❑ HPSS의 사용법은 UNIX의 FTP 사용법과 유사
 - ❖ FTP/HPSS : put/pput, get/pget, mput/mpput, mget/mpget
- ❑ STEP 1. HPSS 계정 신청

```
ykchoi@kisti.re.kr
```

- ❑ STEP 2. HPSS 로그인

```
$ pftp_client core 4021
...
Name (core:u123abc): u123abc
331 Password required for ../../kisti_cell.hpcnet.ne.kr/u123abc.
Password:
ftp>
```

HPSS 사용법 예제 (2/4)

□ STEP 3. Parallel File Store – pput, mpput

ftp> pput <i>file_name</i>	# 단일 파일
ftp> mpput <i>file1 file2 ...</i>	# 다수 파일

```
ftp> pput 300.out
local: 300.out remote: 300.out
200 Command Complete.
200 PORT command successful.
150 Opening BINARY mode data connection for 300.out.
226 Transfer complete.
342635 bytes sent in 0.08 seconds (4.29 Mbytes/s)
```

HPSS 사용법 예제 (3/4)

□ STEP 4. Parallel File Retrieval – pget, mpget

ftp> pget <i>file_name</i>	# 단일 파일
ftp> mpget <i>file1 file2 ...</i>	# 다수 파일

```
ftp> pget 300.out
local: 300.out remote: 300.out
200 PORT command successful.
150 Opening BINARY mode data connection for 300.out (342635 bytes).
226 Transfer complete.
342635 bytes received in 1.62 seconds (205.91 Kbytes/s)
```

HPSS 사용법 예제 (4/4)

STEP 5. 기타 명령어 – del, mdel, bye

```
ftp> del 300.out # 단일 파일 삭제
250 DELE command successful.
```

```
ftp> prompt # 대화(확인) 모드 끄기
Interactive mode off.
ftp> mdel *.out # 다수 파일 삭제
250 DELE command successful.
250 DELE command successful.
...
```

```
ftp> bye # HPSS 종료
421 Timeout (900 seconds): closing control connection.
```

KISTI 병렬형 슈퍼컴퓨터의 주요 소프트웨어 비교표

		IBM Regatta	Compaq	Cray T3E
컴파일러	Fortran	f77, xlf, xlf90, xlf95, xlf_r, xlf90_r, xlf95_r, mpixlf, mpixlf90, mpixlf95, mpixlf_r, mpixlf90_r, mpixlf95_r	f77, f90	f90
	C	cc, xlc, xlc_r, mpcc, mpcc_r	cc	cc
	C++	CC, xlc, mpCC	cxx	CC
디버거		dbx, xldb (GUI)	ladebug, totalview (GUI)	totalview
프로파일러		prof, gprof, xprofiler		pat
수학 라이브러리		BLAS, LAPACK, ScaLAPACK, ESSL, PESSL, IMSL, NAG	BLAS, LAPACK, ScaLAPACK (BLAS, LAPACK은 CXML에 포함)	BLAS, LAPACK, ScaLAPACK, IMSL, NAG

Batch 작업관리 시스템 명령 비교

구 분	IBM Regatta	Compaq	Cray T3E
스케줄러	LoadLeveler	LSF	NQS
Status	llq	bjobs	qstst
Submit	lsubmit	bsub	qsub
Cancel	llhold llcancel	bstop bkill	qdel
info	llclass	bqueues	qstat -b qstat -f name

POE 환경변수 (1/4)

- ❑ **MP_PROCS(-procs):** 태스크 수
 - ❖ 지정하지 않을 경우 기본값 1
 - ❖ CPU 효율을 위해서는 가용 프로세서 개수 이하로 지정
 - ❖ Hybrid(OpenMP+MPI) 프로그램 경우:
 [MP_PROCS x OMP_NUM_THREADS] 또는
 [MP_PROCS x XLSMPOPTS=parthds]
 => 가용 프로세서 개수 넘지 않게 지정
- ❑ **MP_RESD(-resd):** 노드 할당 방식
 - ❖ POE의 직접 노드 할당, 또는 LoadLeveler의 자동 할당 여부 지정
 - ❖ 지정하지 않을 경우 기본값:
 -no: MP_EUILIB=ip, MP_HOSTFILE에 호스트 리스트 파일 지정된 경우
 -yes: 그 외의 경우

POE 환경변수 (2/4)

- ❑ **MP_HOSTFILE(-hostfile, -hfile):** POE의 직접 노드 할당 방식
 - ❖ 호스트 리스트 파일명 명시
 - ❖ 지정하지 않을 경우 기본값: ./host.list
- ❑ **MP_RMPOOL(-rmpool):** LoadLeveler의 자동 노드 선택
 - ❖ 지정된 풀에서의 선택
 - ❖ MP_HOSTFILE=NULL 지정 시 유효함
- ❑ **MP_EUIDEVICE(-euidvice):** 네트워크 인터페이스명 지정
 - ❖ 메시지 패싱 통신을 위한 지정
 - ❖ 네트워크 인터페이스명: en0, css0 등
- ❑ **MP_EUILIB(-euilib):** 통신 모드 지정
 - ❖ ip 또는 us 가능
- ❑ **MP_MSG_API(-msg_api):** 메시지 패싱 라이브러리 설정
 - ❖ MPI, LAPI, MPI LAPI 가능
 - ❖ 지정하지 않을 경우, MPI로 간주

POE 환경변수 (3/4)

- ❑ **MP_LLFILE(-llfile):** LL 작업 명령 파일명 명시
 - ❖ 대화형 POE 작업 수행 시, LoadLeveler 작업 명령 파일에 명시된 키워드 설정에 따라 환경변수 지정
 - ❖ MP_RESD=yes 지정 시 유효함
- ❑ **MP_SHARED_MEMORY(-shared_memory):** 동일 노드 내의 태스크간 메시지 패싱 통신 방식 지정
 - ❖ 공유메모리 사용 또는 실제 네트워크 어댑터 사용 지정
 - ❖ 가능한 값: yes, no
- ❑ **MP_PGMMODEL(-pgmmode):** 병렬 프로그램 모델 명시
 - ❖ Spmd, mpmd 가능
 - ❖ 지정하지 않은 경우 기본값 spmd
- ❑ **MP_CMDFILE(-cmdfile):** 각 태스크가 수행할 프로그램명 리스트 파일명 명시
 - ❖ MP_PGMMODEL=mpmd인 경우 필요

POE 환경변수 (4/4)

❑ MP_INFOLEVEL(-infolevel): 수행 상태 메시지 보고 수준 지정

❖ 기본값 1

❖ 그 외 가능 값

0 : 에러

1 : 경고, 에러

2 : 참고, 경고, 에러

3 : 참고, 경고, 에러 및 IBM 지원센터에서 참고할 진단 메시지

4, 5, 6 : 참고, 경고, 에러 및 IBM 지원센터에서 참고할 상위 및 하위 레벨의 진단 메시지

LoadLeveler 주요 용어 (1/2)

❑ Node

❖ 자체의 운영 체제, CPU 및 메모리를 가지는 독립된 시스템으로 LoadLeveler 클러스터를 구성하는 개별 시스템

❑ Cluster

❖ 작업 부하 분산을 위해 LoadLeveler가 한 묶음으로 관리하는 노드들의 집합

❑ Pool

❖ 클러스터내의 특정 노드들을 편의상 논리적으로 모아 놓은 집합

❑ Class

❖ Pool과 같이 논리적으로 모아 놓은 노드의 집합이며, 가용 자원의 특성, 스케줄링 규칙 등 좀더 구체적인 조건들을 부여한 것

❑ Job Step

❖ 독립적으로 수행되는 작업의 최소 단위

❖ 각 Job Step은 순차 프로그램이거나, 병렬 프로그램임

LoadLeveler 주요 용어 (2/2)

□ Job

- ❖ 한번에 제출하여 수행하는 Job Step들 전체를 의미하며, 종종 하나의 Job Step 의미로도 쓰임

□ Task

- ❖ 하나의 작업에 속한 단일 프로세스로서 운영체제가 부여하는 고유 프로세스 ID를 가지며, 그 자체가 다시 쓰레딩될 수 있음

□ Serial Job

- ❖ 기본 작업 유형 / 하나의 노드에서 수행되도록 짜여진 작업
- ❖ OpenMP는 Serial Job으로 간주

□ Parallel Job

- ❖ 하나 또는 그 이상의 노드에서 여러 개의 Task로 나뉘어 수행되도록 짜여진 작업; 즉 다중 노드 및 CPU 작업을 말함
- ❖ LoadLeveler는 PVM과 MPI의 2가지 병렬 작업을 지원함

LoadLeveler 작업 명령 키워드 (1/4)

□ job_type: 작업 유형 지정

- ❖ serial, parallel, pvm3 가능
- ❖ 명시하지 않을 경우 serial로 간주

□ job_name: 작업 이름 지정

- ❖ 명시하지 않을 경우 LoadLeveler가 임의로 부여

□ step_name: 작업 파일 내에 Job Step 이름 지정

- ❖ 명시하지 않을 경우 LoadLeveler에 의해 작업 명령 파일 내의 순서대로 부여(0, 1, 2, ...)

□ input: standard input 지정

□ output: standard output 지정

□ error: standard error 지정

□ initialdir: 초기 작업 디렉토리 지정

LoadLeveler 작업 명령 키워드 (2/4)

- ❑ **node:** Job Step을 위해 할당할 노드 수 지정
 - ❖ 각 Job Step 마다 지정해 주어야 함
 - ❖ 노드 수를 하나의 숫자로 지정하거나 범위로 지정할 수 있음
 - ❖ 지정하지 않을 경우 1로 간주
 - ❖ **tasks_per_node** 또는 **total_tasks** 키워드와 함께 사용
- ❑ **blocking:** 노드 당 태스크 분배 방법
 - ❖ 노드 당 일정한 개수씩 태스크 분배
 - ❖ 한번에 분배할 태스크 개수
 - ❖ 양의 정수 또는 **unlimited** 가능
- ❑ **tasks_per_node:** 노드 당 태스크 수 지정
- ❑ **total_tasks:** 수행할 전체 태스크 수 지정
- ❑ **class:** LoadLeveler class 지정
 - ❖ 지정하지 않을 경우 **No_Class**라는 이름의 클래스로 간주

LoadLeveler 작업 명령 키워드 (3/4)

- ❑ **node_usage:** 여러 작업 수행 허용 여부 지정
 - ❖ 해당 Job Step 수행 시, 동일 노드 내에 다른 Job 수행 허용 여부 지정
 - ❖ **Shared, not_shared** 가능
- ❑ **notification:** 알림 메일을 보낼 조건 지정
 - ❖ **always, error, start, never, complete** 가능 / **complete** 주사용
 - ❖ 지정하지 않을 경우 **complete**로 간주
- ❑ **notify_user:** 알림 메일을 보낼 주소 지정
 - ❖ 지정하지 않을 경우, 작업 제출 노드의 작업 제출자 주소로 메일이 보내짐
- ❑ **resources:** consumable resource 종류 및 요구량 지정
- ❑ **requirements:** 할당될 노드가 만족해야 할 필수 요구 조건 지정
- ❑ **wall_clock_limit:** 예상 작업 소요 시간(Elapsed time) 지정
 - ❖ 초단위 또는 시:분:초 형식으로 지정

LoadLeveler 작업 명령 키워드 (4/4)

□ **network:** 통신 프로토콜, 네트워크 어댑터, 통신 특성 지정

❖ 각 **Job Step**마다 명시

❖ 지정 형식:

```
#@ network.protocol = network_type [, [usage] [, mode [, comm_level] ] ]
```

❖ **protocol:** MPI, LAPI, PVM 가능

❖ **network_type:** 반드시 명시

- 어댑터 이름(en0, csss 등) 또는 인터페이스 종류(Etherchannel, ethernet, switch 등)로 나타낼 수 있음

❖ **usage:** shared 또는 not_shared 지정 / 지정하지 않을 경우 shared

❖ **mode:** IP 또는 US

- 지정하지 않을 경우 IP로 간주

- **US mode:** SP 스위치 **network_type**에서만 지원됨

❖ **comm_level :** 태스크간 통신량 수준 명시

- **US mode**에서만 지원됨

- **LOW, AVERAGE, HIGH** 가능