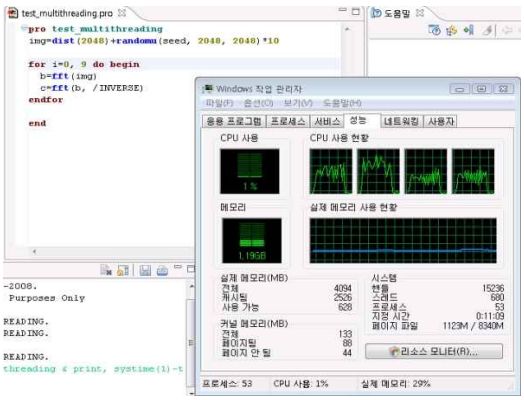
IDL이 쿼드코어도 지원하나요?

그럼요, Quad 반 Dual 반의 시대 아닙니까?

2008년 "쿼드코어 CPU를 IDL이 지원하는가"에 대한 질문을 많이 받았습니다. 그만큼 이제는 쿼드코어 CPU가 대중화 되었다는 의미이기도 합니다. PC 방에서도 쓰는 흔한 CPU인데, IDL이 지원 못하면 안되겠지요.

그러므로 답변은, "네, 잘 지원합니다. 쿼드코어의 성능을 잘 활용할 수 있습니다" 입니다. 쿼드코어 Xeon CPU 네개를 사용하는 -16 코어네요- 시스템에서 좋은 효과를 본 경험도 있습니다. 운영체제들이 잘 지원을 하고 있으므로, IDL 사용자들은 크게 신경 쓸 일 없이 여러 CPU를 사용하는 컴퓨터를 활용할 수 있을 것입니다.

다음 그림은 쿼드 코어 CPU와 Windows Vista를 사용하는 시스템에서 FFT를 수행할 때에 네 개의 코어에 작업이 나뉘어 걸리는 모습입니다.



I네 개의 CPU를 사용하여 FFT 계산

사용자가 어떻게 해 주어야 하나요?

그냥 사용하면 됩니다. IDL이 처음 시작될 때, 운영체제로부터 CPU 개수를 인식하게 됩니다. 처리할 데이터의 분량을 보아 다중 CPU를 활용하는 것이 효율적이라면, IDL이 알아서 CPU에 일을 분배합니다. 사용자는 IDL의 기본 사용법인 "배열기반의 연산"에만 충실하게 프로그램을 작성하면 됩니다. 이처럼 작업을 여러 갈래로 나누어 처리하는 것을 "멀티 쓰레딩(Multi-Threading)"이라고 하는데요, IDL의 모든 기능이 멀티 쓰레딩을 지원하는 것은 아니지만, 연산 대상이 되는 데이터 개수가 많은 작업은 대부분 지원하고 있습니다. 예를 들면 1000X1000 이미지의 FFT 변환 같은 작업 말이죠.

IDL이 CPU 개수를 어떻게 알아내나요?

IDL이 시동되는 순간, IDL은 운영체제로 부터 가용할 수 있는 CPU의 개수를 알아냅니다. 그 결과를 !CPU 시스템 변수에 저장해 둡니다.

```
IDL> help, !cpu, /struct
  HW_VECTOR       LONG      0
  VECTOR_ENABLE   LONG      0
  HW_NCPU         LONG      8
  TPOOL_NTHREADS  LONG      8
  TPOOL_MIN_ELTS  LONG64    100000
  TPOOL_MAX_ELTS  LONG64    0
```

HW_NCPU 필드에 사용 가능한 CPU 개수가 기록되어 있습니다. 위 예는 쿼드 코어 CPU를 두개 사용하는 시스템을 본 것입니다. HW_NCPU는 8개이므로, 기본값으로 TPOOL_NTHREADS(실행 쓰레드의 개수)도 8개로 잡힙니다. TPOOL_MIN_ELTS(멀티쓰레드를 가동하기 위한 최소 데이터 개수)는 기본적으로 10만개로 잡혀 있습니다. TPOOL_MAX_ELTS는 0으로 잡혀있는데, 이는 무제한을 의미합니다. 데이터의 개수가 MIN_ELTS보다 적거나 MAX_ELTS보다 많으면(너무 적거나 너무 많으면) 멀티쓰레딩을 가동하지 않습니다. 그럴 만한 이유가 있습니다.

역효과가 날 수도 있어요!

멀티쓰레딩이 만능은 아닙니다. 다음과 같은 상황에서는 오히려 더 느려질 수 있습니다.

1) 별 일 아닌데도 일을 분배하려는 경우

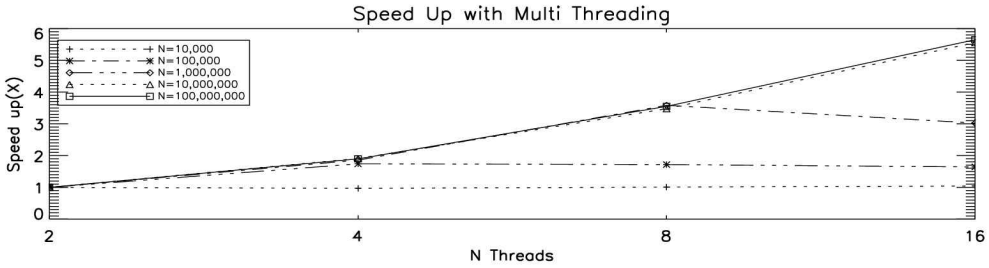
일을 분배하고, 나중에 결과를 취합하는 데에 시간이 더 걸릴지도 모릅니다.

2) 너무 많은 양의 데이터를 분배하여 처리하려는 경우

각각의 쓰레드는 메모리(RAM)을 차지합니다. 이 때, 물리적인 메모리(RAM)가 모자란다면 운영체제는 하드디스크를 메모리처럼 이용하는 가상 메모리를 사용하게 됩니다. 하드디스크 억세스 속도와 메모리 억세스 속도는 하늘과 땅차이입니다.

3) 여러 프로세스가 CPU 사용권을 다투게 되는 경우

거의 모든 경우, 컴퓨터에서 IDL만 실행하고 있는 것은 아닙니다. 연산 시간이 긴 프로그램을 실행하면서 기다리는 동안 다른 일을 할 수도 있습니다. 다른 사용자가 프로그램을 실행시키고 있는 중일



Multi Threading에 의한 처리 속도 향상

수도 있습니다. 프로세스들이 CPU를 번갈아 사용하는 상황이라면, 차라리 쓰레드 수를 줄이더라도 CPU를 꾸준히 IDL 연산에 사용할 수 있는 게 나을 수도 있습니다.

이런 이유로 사용자가 쓰레드 수, 쓰레드 가동 최소/최대 데이터 수를 설정할 수 있어야 합니다.

멀티쓰레드의 컨트롤

일반적으로는 IDL의 기본 설정대로 사용하는 것이 무난합니다만, 앞의 예와 같이 역효과가 나타나는 경우 !CPU의 설정을 바꾸거나, 실행함수의 키워드를 통해 멀티쓰레드를 제어할 수 있습니다.

1) !CPU의 설정을 바꾸는 방법

!CPU 시스템 변수는 읽기 전용이므로, 구조체 내의 필드값을 직접 수정할 수 없습니다. CPU 프로시저를 사용하면 !CPU 구조체의 설정값을 변경할 수 있게 됩니다.

예) IDL> cpu, tpool_nthreads=4

2) 함수의 키워드를 이용하는 방법

어떤 함수들을 도움말에서 찾아 보면 Thread Pool Keywords를 사용할 수 있다는 설명을 볼 수 있습니다. 키워드를 사용하면, !CPU의 값보다 우선하여 키워드 값에 따라 작동을 하게 됩니다.

예) IDL> x=cos(r, tpool_min_elts=10000)

멀티쓰레드 효과 분석

	10^4개	10^5개	10^6개	10^7개	10^8개
1 Thd	0.00121	0.01288	0.13937	1.41079	14.1202
2 Thds	0.00125	0.00740	0.07500	0.73871	7.43789
4 Thds	0.00120	0.00750	0.03890	0.40570	3.97754
8 Thds	0.00116	0.00784	0.04068	0.25387	2.49942

데이터 개수와 쓰레드 개수에 따른 처리 시간(단위:초)

위 표는 10^4개, 10^5개, 10^6개, 10^7개, 10^8개의 요소를 가지는 배열 변수 data에 대해, 다음 식을 수행하는 데 걸리는 시간을 보여 줍니다.

result=sqrt(cos(data*!dtor)^2+sin(data*!dtor))

사용된 장비는 Intel Xeon Quad Core 3.2Ghz 2개, 16GB 메모리를 사용하는 64bit Linux 시스템입니다.

TPOOL_MIN_ELTS가 100000으로 설정되어 있으므로, 10,000개의 데이터에서는 멀티쓰레딩이 작동하지 않습니다. 처리 속도가 모두 같습니다. 100,000개의 경우는 1 쓰레드 보다 2 쓰레드가 더 빨라집니다만, 그보다 더 많은 쓰레드를 생성해서 더 빨라지는 것 같지는 않습니다. 10^6개인 경우에는 4쓰레드까지는 효과가 나타나는 것 같습니다. 10^7개나 10^8개는 8쓰레드까지 효과가 잘 나타나고 있습니다.

쓰레드 풀(Thread Pool) 사용 조건 정리

1) CPU 2개 이상

하나의 CPU인 시스템에서 멀티쓰레딩을 설정해 봐야 효과가 없습니다. 다음 메시지는 8개의 CPU인 시스템에서 16개의 쓰레드를 실행하려는 데에 대한 IDL의 부정적인 경고 메시지입니다. 이런 경우 처리 속도는 조금 더 느려질 뿐입니다.

% CPU: Warning: Using more threads (16) than the number of CPUs in the system (8) will degrade performance.

2) 사용하는 명령어나 연산자가 멀티쓰레딩을 지원

거의 모든 배열처리 명령어, 함수, 연산자가 멀티쓰레딩을 지원합니다. 온라인 도움말에서 "Routines that use the thread pool"로 검색을 하면 멀티쓰레딩 지원 연산자와 루틴들의 목록을 볼 수 있습니다.

3) 처리하고자 하는 데이터의 개수가
TPOOL_MIN_ELTS < N < TPOOL_MAX_ELTS
의 조건을 만족

결론

CPU가 많은 컴퓨터를 사용하는 것은 좋은 일입니다. IDL 사용자들은 따로 코드를 수정하거나 조치를 취할 필요가 거의 없습니다.