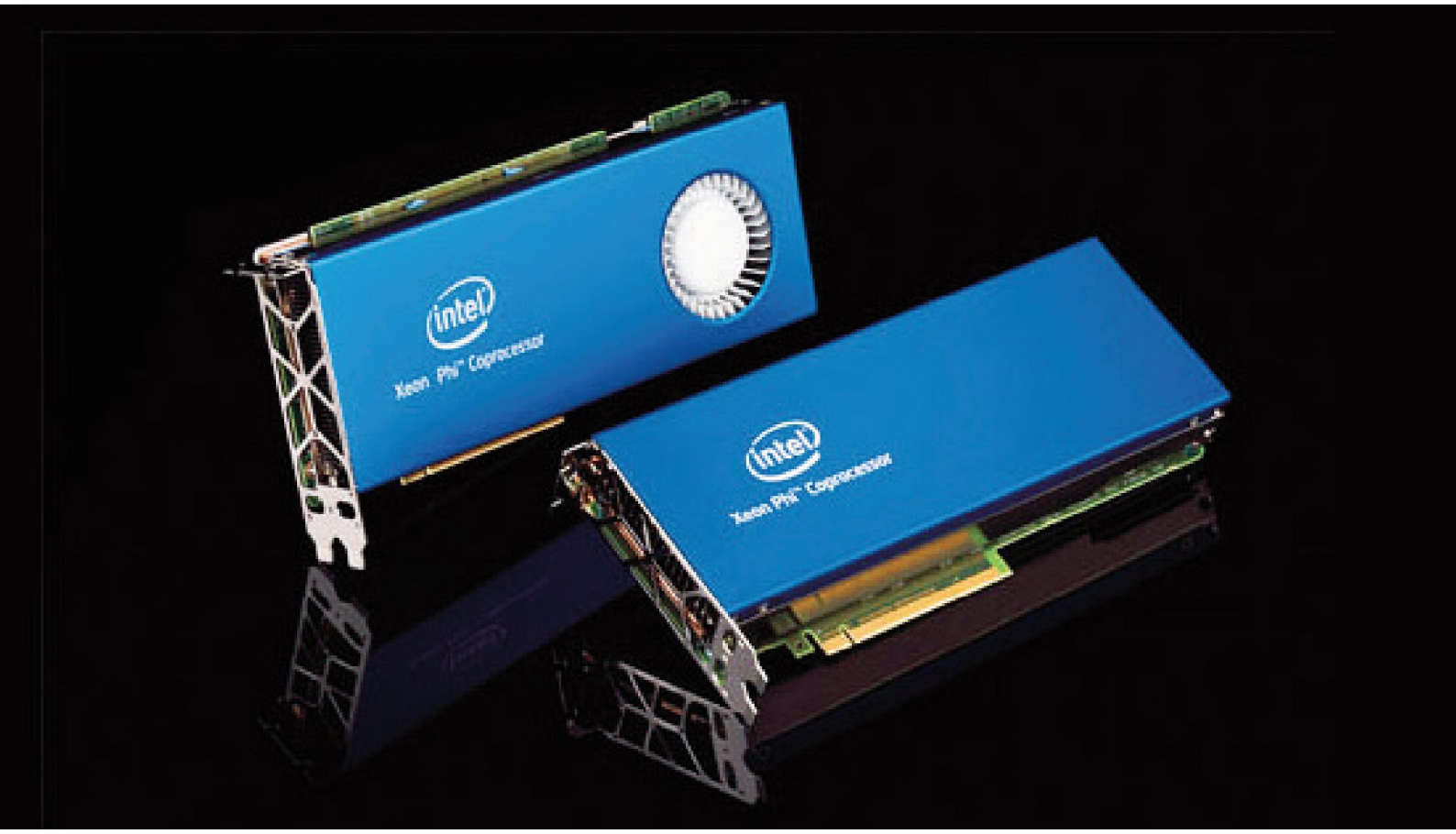
슈퍼컴퓨팅 산업 : 최신 가속기 활용 1

Intel® Xeon Phi™ 코프로세서 소개

이성호 이사 (인텔코리아 ESS 기술담당)



Intel® Xeon Phi™ 코프로세서란?

Intel® Xeon Phi™ 는 Intel® Many Integrated Core 아키텍쳐(Intel MIC 아키텍쳐) 기반으로 새롭게 출시된 코프로세서(coprocessor)이다. 2012년 말에 1세대 제품이 정식 출시되었으며 코프로세서라는 명칭이 의미하듯이 기존에 범용으로 사용되던 Intel® Xeon® 프로세서 제품과 함께 사용되어 애플리케이션 처리 성능을 향상시켜주는 역할을 한다. 특히, 물리, 생물, 화학 및 금융분야 등에서 고도로 병렬화 된 작업부하 처리시 뛰어난 성능을 보여줄 수 있도록 설계되었다. 또한 기존 Intel® Xeon® 프로세서 환경에서의 표준화된 프로그래밍 방법을 그대로 사용할 수 있기 때문에 개발자들이 손쉽고 빠르게 병렬 코드를 작성하고 최적화 할 수 있게 해준다.

Intel® Xeon Phi™는 FORTRAN, C, C++등의 범용 프로그래밍 언어를 지원하고 있으며x86 메모리 모델과 부동소수점 표현을 위한 IEEE 754표준을 지원한다. 또한 기존의 x86기반의 각종 라이브러리나 컴파일러, Intel® VTune™ 같은 개발 툴을 그대로 활용할 수 있다는 장점을 가지고 있다.

Intel기반 프로세서 제품 군의 병렬처리 환경

개발자들이 본격적으로 높은 수준의 병렬처리에 관심을 갖게 됨에 따라 인텔에서도 일반적으로 잘 알려지고 이해가 쉬운 프로그래밍 모델을 기반으로 병렬처리를 위한 아키텍처를 제공하고 있다. 그 결과 프로그래머들은 CPU를 위한 알고리즘을 작성하고 이를 인텔 MIC 아키텍쳐에 확대 적용할 수 있게 되었다. 인텔 CPU에서의 성능 최적화를 위한 기법들, 예를 들면, 여러 개의 코어나 쓰레드를 이용할 수 있도록 애플리케이션을 확장하거나, 계층구조의 메모리와 캐시의 활용을 위해 데이타를 차단(blocking)하는 것, 또 SIMD(Single Instruction Multiple Data)의 효과적인 사용 등과 같은 기법들이 인텔 MIC에서도 성능 극대화를 위해 그대로 적용할 수 있다.

인텔 MIC 아키텍쳐는 인텔 CPU의 논리적인 확장 판이라 할 수 있다. 그 결과 병렬처리를 위한 CPU 코드의 MIC에 대한 대폭적인 재사용을 통해 프로그래머의 생산성을 향상시키고 코딩 및 성능 향상을 위해 필요한 시간을 대폭 감소시킬 수 있게 되었다.

	64-bit Intel® Xeon® processor	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor E5 Product Family	Aubrey Isle (in Knights Ferry)	Intel®Xeon Phi™ coprocessor (Knights Corner)
Frequency	3.6GHz	3.0GHz	3.2GHz	3.3GHz	2.7GHz	1.2GHz	~1.1GHz
Core(s)	1	2	4	6	8	32	Up to 61
Thread(s)	2	2	8	12	16	128	Up to 244
SIMD width	128(2 clock)	128(1 clock)	128(1 clock)	128(1 clock)	256(1 clock)	512 (1 clock)	512 (1 clock)

그림 1. 인텔 프로세서 제품별 병렬처리 환경

그림 1에서 보듯이 최근까지 Intel® Xeon® 프로세서 기반으로 계속적인 성능향상을 통해 병렬처리를 지원해 왔다. 멀티코어 환경으로 전환에 따라 클럭스피드는 상대적으로 낮아졌으나 코어 개수는 계속적으로 증가해 왔음을 알 수 있다. 최신의 E5 제품 군(코드명 Sandy Bridge)의 경우 프로세서당 8개까지의 코어를 지원하고 있으며 올 하반기 출시예정인 E5 v2제품군(코드명 Ivy Bridge)의 경우 12개까지의 코어를 제공할 예정이다. 하지만 고도로 병렬화된 작업의 보다 신속한 처리를 위해서 Intel® Xeon Phi™ 코프로세서를 개발하게 되었고, 초기 프로토타입인 Knight Ferry를 통한 베타 테스트를 거쳐 정식제품인 Intel® Xeon Phi™를 2012년 말 출시하게 되었다. 국내에서는 KISTI가 베타 테스트 사이트로 지정되어 정식 제품 출시에 큰 역할을 하였다.

Intel® Xeon Phi™ 코프로세서와 Intel® Xeon® 프로세서의 비교

Intel® Xeon® 프로세서는 멀티코어 아키텍쳐 기반으로 설계된 범용 프로세서이다. 따라서, 다양한 유형의 HPC 애플리케이션뿐만 아니라 일반적인 기업용 애플리케이션 등에도 뛰어난 성능을 보여준다. 특히, 요즈음 화두가 되고 있는 전력효율 측면에서도 뛰어난 와트당 성능 (performance/watt)을 제공하고 있다.

Intel® Xeon Phi™는 인텔 MIC 기반으로 설계되었으며, 고도로 병렬화되고 연산이 많은 작업처리를 위해 최적화되어 있다. 범용의 프로그래밍 언어 및 각종 소프트웨어 개발 툴들을 Xeon® 프로세서와 공유할 수 있어서 신속하고 효과적인 애플리케이션 개발 및 성능 튜닝이 가능하다. 현재 출시된 제품은 22나노미터 공정으로 설계되었으며 50개 이상의 코어를 제공하고 있다.

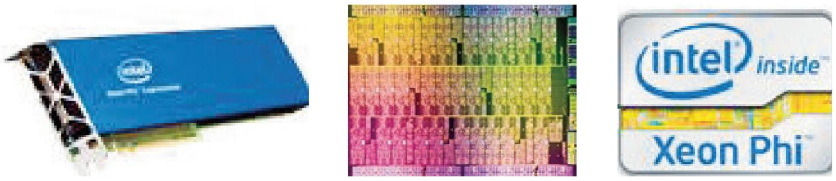


그림 2. Intel® Xeon Phi™ 코프로세서 - 외관, 현미경 사진 및 로고

Intel® Xeon Phi™코프로세서를 사용하기 위한 플랫폼 환경

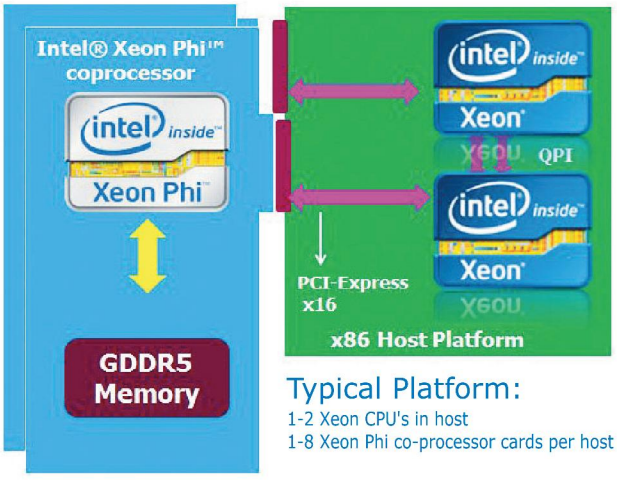


그림 3. 전형적인 Intel® Xeon Phi™ 코프로세서 사용 환경

Intel® Xeon Phi™ 코프로세서와 Intel® Xeon® 프로세서는 동일한 플랫폼, 즉 서버 내에서 사용된다. 이러한 서버들을 인터커넥트를 통해 클러스터 혹은 슈퍼컴퓨터 형태로 구성할 수 있다. 현재는 Intel® Xeon Phi™ 코프로세서만으로 독자적인 시스템을 구성할 수 는 없다. 독자적으로 구동하는 제품도 향후 출시를 검토 중이다. 플랫폼 내의 Intel® Xeon® 프로세서는 캐시일관성(cache coherence)을 유지하며 프로세서간에 메인 메모리를 공유한다. Intel® Xeon Phi™ 코프로세서는 캐시일관성을 제공하는 SMP-on-a-chip 디바이스이며 다른 디바이스와 PCIe버스를 통해 연결되는데, 동일한 노드나 시스템상에서 Intel® Xeon® 프로세서나 다른 Intel® Xeon Phi™ 코프로세서와의 하드웨어 캐시 일관성을 제공하지는 않는다.

Intel® Xeon Phi™ 코프로세서는 리눅스상에서 구동된다. 리눅스를 사용하기 위한 하나의 진정한 의미의 x86 SMP-on-a-chip이라 할 수 있다. 각각의 Intel® Xeon Phi™ 카드는 독자적인 IP주소를 가진다. 시스템에 터미널 윈도우로 로그온 하게 되면 먼저, Intel® Xeon® 프로세서 기반 호스트의 shell로 접속하게 된다. 그런 다음 "ssh mic0" 명령을 실행하면 첫번째 Intel® Xeon Phi™ 코프로세서 카드로 로그온 할 수 있다. 여기에서 "cat /proc/cpuinfo"를 실행하게 되면 6100줄의 정보가 표시된다. 다음에 있는 그림 4는 내용 중 일부를 보여준다. 이와 같이 Intel® Xeon

Phi™ 코프로세서는 기존 Intel® Xeon® 프로세서 환경과 매우 유사하여 친숙함을 느낄 수 있을 것이다.

윈도우 상에서 "ssh"명령이나 "emacs", 혹은 "vi"를 실행할 수 있으며, "awk"스크립트나 "perl"을 실행할 수도 있다. MPI코드를 코프로세서 카드상의 각 코어에서 실행할 수도 있도록 start할 수도 있고, 다른 컴퓨터와 접속하여 동시에 처리하는 것도 가능하다.

그림 4에서 보면 processor 번호가 243으로 되어 있음을 볼 수 있는데, 이 것은 61개의 코어가 각각 4개씩의 쓰레드를 가지고 있기 때문에 쓰레드가 0부터 243까지, 즉 244개가 제공된다는 의미이다.

그림 4. Intel® Xeon Phi™ 코프로세서 " cat /proc/cpuinfo" 실행 화면

Intel® Many Integrated Core 아키텍쳐

프로그래밍을 함에 있어서 디바이스에 대한 깊은 지식이 꼭 필요하지는 않지만, 코프로세서의 여러 특성을 파악할 수 있다면 많은 도움이 된다. 프로그래밍 관점에서 보면, 코프로세서를 코어당 여러 개의 쓰레드를 가진, 50개 이상의 코어로 구성된 512비트 SIMD명령어를 지원하는 하나의 x86기반 SMP-on-a-chip으로 볼 수 있다는 것이 핵심이다.

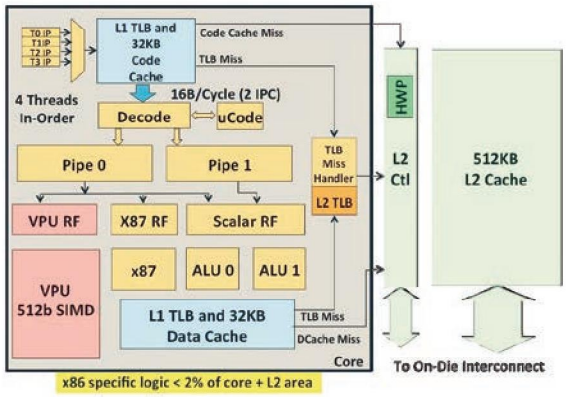


그림 5. Intel® Xeon Phi™ 코프로세서 코어 내부 구조

코어는 순차(in-order) 처리 기반의 x86 프로세서 코어이며 예전의 펜티엄 프로세서의 설계와 유사하다. 하지만, 64비트 지원과 함께 코어당 4개의 쓰레드, 전력관리기능, 링(ring)형태 인터커넥트, 512비트 SIMD기능 지원 및 추가로 향상된 여러 기능들로 인해 20여년 전의 펜티엄 코어와는 확연한 차이를 보이고 있다. x86 특유의 논리회로(L2 캐시는 제외)는 Intel® Xeon® Phi™ 코프로세서 다이(die) 영역의 2% 미만을 차지하고 있다.

다음은 최근에 정식제품으로 출시된 Intel® Xeon Phi™ 코프로세서 제품과 관련된 세부사항이다.

- 2012년 말에 정식제품으로 출시된 코프로세서 제품이며 시스템상에서 적어도 하나의 프로세서를 필요로 함
- 리눅스 기반에서 작동 (http://www.intel.com/software/mic 에서 소스코드 제공)
- 인텔의 22나노미터 공정과 3D 트라이게이트 트랜지스터 기술로 제조
- Intel Cluster Studio XE 2013을 포함한 표준 개발도구 지원 (http://www.intel.com/software/mic 참조)
- Many Core지원
  - 50개 이상의 코어 지원 (특정 코어 개수를 지정하는 형태의 하드코딩 애플리케이션은 피하는 것이 좋음)
  - 순차(in-order) 코어로서 64비트 x86 명령어를 지원하며 향상된 SIMD 기능 지원

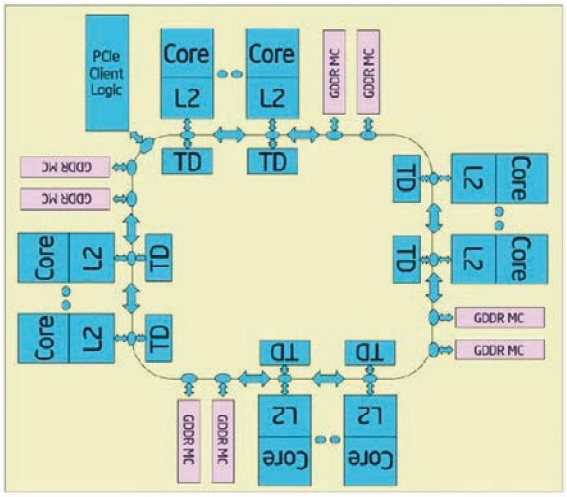


그림 6. 인텔 MIC 아키텍쳐

■ 코어당 4개의 하드웨어 쓰레드(하나의 디바이스에서 200개 이상의 하드웨어 쓰레드 지원)를 통해 in-order 마이크로아키텍쳐의 단점인 지연현상을 없애줌. 따라서 애플리케이션에서 Intel® Xeon® 프로세서의 하이퍼쓰레딩 기능보다 Intel® Xeon Phi™ 코프로세서에서 제공하는 하드웨어 쓰레드를 사용하는 것이 더욱 효과적임

■ 코어간에는 고속의 양방향 링(ring)으로 연결됨

■ 코어의 클럭스피드는 1GHz이상

■ 모든 프로세서에 걸쳐 캐시 일관성(cache coherence) 제공

■ 각 코어는 512KB L2 캐시를 가지며 다른 코어의 L2 캐시에 고속으로 액세스 가능 (총 25MB)

■ 캐시는 높은 대역폭을 가진 메모리를 제공함과 동시에 높은 전력효율을 제공

– 64비트 x86 명령외에 추가된 특별 명령어

■ 기존의 MMX™, Intel® SSE, 혹은 Intel® AVX 대신 512비트 폭의 벡터를 통해 더욱 향상된 SIMD기능 제공

■ reciprocal, square root, power 및 exponent 연산을 위한 고성능 제공

■ 고효율 메모리 대역폭을 얻기 위한 scatter/gather 및 streaming store 기능

– 특별한 기능

■ 패키지 내부 메모리 컨트롤러를 통해 최대 8GB용량의 GDDR5지원

■ PCI-e 연결 로직이 on-chip형태로 제공

■ 전력관리 기능 제공

■ Intel® VTune™ Anplifier XE2013 같은 개발 툴을 위한 성능 모니터링 기능 제공

Intel® Xeon Phi™ 와 GPU의 차이점

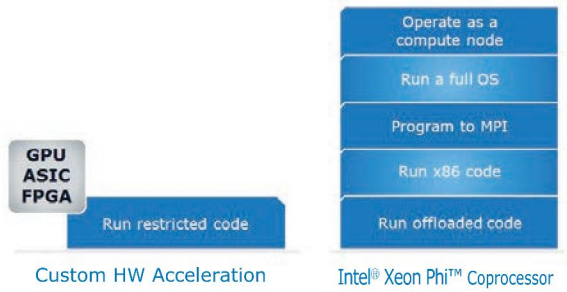


그림 7. GPU 와 Intel® Xeon Phi™ 코프로세서

양쪽 모두 벡터라이제이션 혹은 대역폭과 결합된 스케일링에 의해 가속화 할 수 있는 코드의 서브셋을 공유할 수 있다. 달리 말하면, GPU에서 우수한 결과를 보여주는 애플리케이션은 Intel® Xeon Phi™ 에서도 항상 좋은 결과를 보여준다. 왜냐하면 동일한 벡터라이제이션 혹은 대역폭이 반드시 존재하기 때문이다. 반대의 경우는 성립하지 않는다. 또한, GPU의 경우는 Intel® Xeon Phi™ 가 제공하는 형태로 프로그램을 작성할 수 가 없다. Intel® Xeon Phi™ 코프로세서는 GPU에서는 실행할 수 없는 애플리케이션도 지원하는 유연성을 제공한다. 이러한 점이 Intel® Xeon Phi™ 가 내장된 시스템이 GPU를 사용하는 시스템보다 좀더 광범위한 활용도를 갖게 되는 이유이다. 추가로, GPU를 위한 튜닝이 일반적으로 많은 노력과 시간을 필요로 하는데 비해 Intel® Xeon Phi™ 코프로세서의 경우는 상대적으로 훨씬 쉽게 접근

할 수 있고 현재와 미래의 플랫폼에 쉽게 적용할 수 있는 애플리케이션 환경을 구축할 수 있다는 장점을 갖고 있다.

변환 및 튜닝의 이중 효과

프로그래밍이란 쉬운 작업이 아니다. 하물며 병렬 프로그래밍은 더욱 어려운 것이 사실이다. 그렇지만 어쨌든 병렬 연산을 최대화하고 데이터의 이동을 최소화하여야 한다는 기본을 지키기 위해 계속 노력을 해야만 한다. 병렬 연산은 더 많은 코어와 쓰레드에 대한 스케일링 및 한 번에 더 많은 데이터를 처리하기 위한 벡터 처리에 의해 활성화 된다. 데이터 이동의 최소화는 알고리즘 측면의 노력을 필요로 하지만 Intel® Xeon Phi™코프로세서에 적용된 인텔 MIC 아키텍처에서 제공하는 메모리와 코어간의 더 넓은 대역폭을 통해 보다 쉽게 처리할 수 있다. 아울러 인텔 기반 제품 전반에 걸쳐 병렬 프로그래밍을 위해 동일한 프로그래밍 언어와 모델을 사용할 수 있다. Intel® Xeon Phi™코프로세서는 Fortran, C, C++같은 언어를 완벽하게 지원하며, 보편적으로 이용되는 프로그래밍 모델인 OpenMP, MPI 및 인텔의 TBB 등도 완벽하게 지원된다. 널리 확산되고 있는 새로운 모델인 Coarray Fortran, Intel Silk™ Plus 및 OpenCL등도 적용이 가능하다.

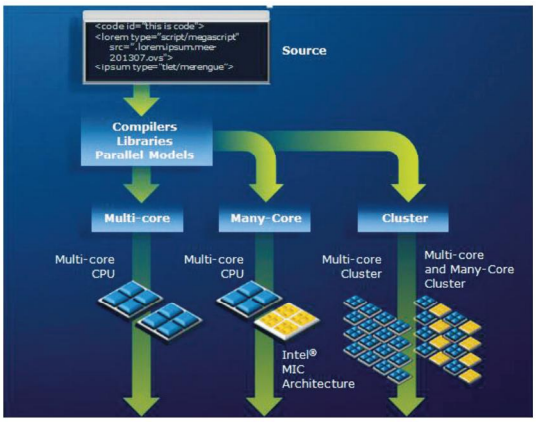


그림 8. 변환 및 튜닝의 이중 효과

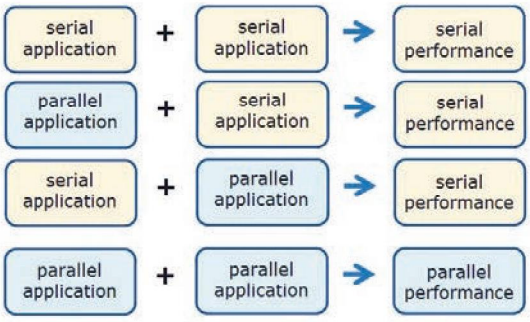


그림 9. 고성능 구현을 위한 환경의 조합
- 병렬 소프트웨어 + 병렬 하드웨어

스케일링, 벡터 사용, 그리고 메모리 사용을 위해 Intel® Xeon Phi™ 코프로세서상에서 튜닝 한 부분은 Intel® Xeon® 프로세서상에서 애플리케이션을 실행할 경우에도 역시 성능의 향상을 기대할 수 있으며, 애플리케이션이 Intel® Xeon Phi™ 코프로세서상에서 최고의 성능구현에 실패했다면 일반적으로 스케일링, 벡터 사용, 혹은 메모리 사용 부분으로 돌아가서 추적하면 해결의 실마리를 찾을 수 있다. 이러한 문제가 제기 되었을 때 발견한 해결책은 Intel® Xeon® 프로세서에서 실행시에도 그대로 활용할 수 있다.

Intel® Xeon Phi™코프로세서 환경의 성능 극대화 방법

Intel® Xeon Phi™코프로세서를 제대로 활용하기 위한 가장 좋은 방법은 먼저 애플리케이션이 Intel® Xeon® 프로세서에서 얻을 수 있는 성능 향상 방법을 충분히 찾아내는 것이다. Intel® Xeon® 프로세서상에서 병렬처리 기능을 극대화 하지 않은 상태에서 Intel® Xeon Phi™ 코프로세서를 사용해 본다면 실망스런 결과를 얻게 될 것이 거의 확실하다. 그림 9는 핵심적인 사항을 보여준다. 더 나은 성능의 구현은 병렬화된 애플리케이션과 병렬처리용 하드웨어의 조합이 만들어 질때 가능하다.

100개 이상의 쓰레드 스케일링의 중요성

애플리케이션이 Intel® Xeon Phi™ 코프로세서상에서 사용할 수 있도록 준비가 되었다면, 스케일링만큼 중요한 것이 없다고 해도 과언이 아니다. 애플리케이션이 100개 이상의 쓰레드로 스케일 가능하여야 고도로 병렬화가 되었다고 할 수 있다. 벡터와 메모리 대역폭의 효과적인 사용 또한 매우 중요하다. 작업, 쓰레드, 벡터 등등에 대한 고도의 병렬화를 활용하도록 작성되지 않았거나 변경되지 않은 애플리케이션은 고도의 병렬 도를 제공하도록 설계된 하드웨어로부터의 이점을 제대로 활용할 수 없다.

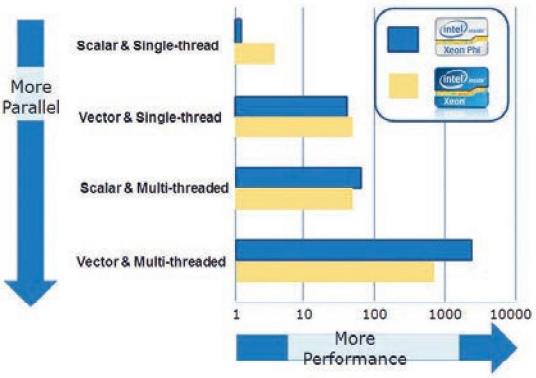


그림 10. 쓰레드와 벡터 작업부하를 결합한 경우

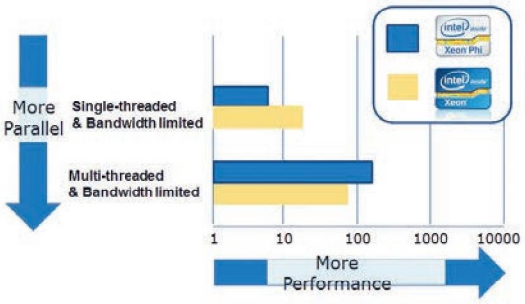


그림 11. 메모리 사용이 많은 작업 부하

그림 10과 11은 여러 유형의 애플리케이션이 Intel® Xeon® 프로세서와 Intel® Xeon Phi™ 코프로세서상에서 두가지 주요 케이스(연산 위주와 메모리 위주 애플리케이션)에 대해 어떻게 반응하는지를 보여주는 사례이다. 로그자(logarithmic scale)가 그래프에 채택되었다. 따라서 아래에 있는 성능 표시 막대는 위쪽에 있는 표시 막대에 비해 대폭적인 성능 향상을 보여준다. 결과는 애플리케이션에 따라 달라진다. 애플리케이션에 의해 현재의 벡터와 쓰레드 사용량 및 합쳐진 대역폭의 측정을 통해 애플리케이션의 어떤 부분이 고도로 병렬화 된 하드웨어에 적용할 수 있는지를 파악할 수 있다. "더 많은 병렬화"를 통해 프로세서와 코프로세서 양쪽 모두의 성능을 향상 시킬 수 있다. 애플리케이션에 대한 "더 많은 병렬화"는 Intel® Xeon® 프로세서와 Intel® Xeon Phi™ 코프로세서 모두의 장점을 활용 할 수 있는데, 그 이유는 이들 모두 범용 프로그래밍이 가능한 디바이스이기 때문이다.

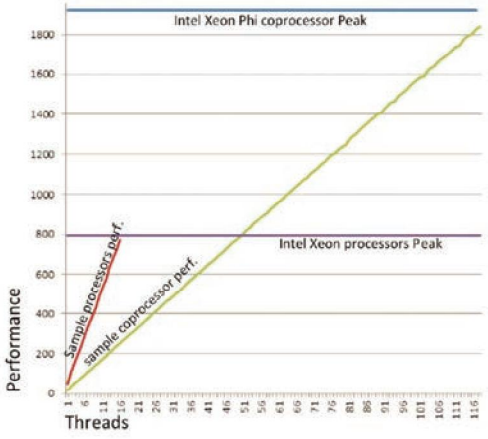


그림 12. 성능과 병렬화의 상관관계

다음 그림 12, 13과 14에서는 많은 수의 쓰레드와 벡터 사용의 필요성을 보여준다. 그림 12에서는 Intel® Xeon Phi™ 코프로세서가 Intel® Xeon® 프로세서를 넘어서는 수준의 성능 구현이 가능하다는 사실을 보여준다. 다만, 그렇게 하기 위해서는 더 많은 병렬화가 필요하다. 그림 13과 14에서는 몇 가지 중요한 사항을 지적하기 위해 그림 11의 일부를 좀 더 자세히 보여준다. 그림 13은 높은 수준의 병렬화에 최적화된 하드웨어(여기서는 Intel® Xeon Phi™)에서 제공하는 수준의 성능에 도달하기 위한 더 많은 병렬화에 대한 전반적인 필요성을 보여준다. 그림 14에서는 Intel® Xeon® 프로세서에서 제공하는 최고 수준 정도로 "병렬화"를 제한하는 것은 Intel® Xeon Phi™ 코프로세서에 흥미를 갖게 만들기에는 충분하지 못하다는 것을 알수있다. 그림 13과 14는 그림 12가 보여주고자 하는 점의 핵심을 좀더 이해가 쉽도록 보여준다.

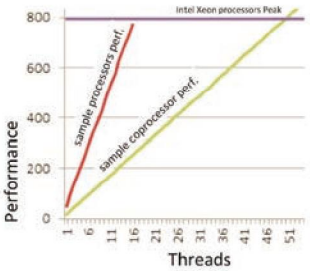


그림 13. Intel® Xeon® 제공 수준 성능으로 제한

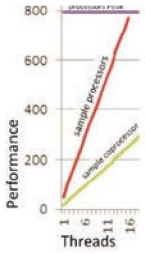


그림 14. Intel® Xeon® 제공 수준 쓰레드로 제한

100개 이상의 많은 쓰레드를 얻는 방법

다행히도 OpenMP, Fortran의 do concurrent, Intel® Threading Building Blocks (TBB) 및 Intel® Cilk™ 등을 사용하면, 프로그램상에서 많은 쓰레드를 작성할 수 있다. 일반적으로 제일 바깥에 있는 루프는 directive, keyword혹은 template등에 의해 많은 쓰레드를 사용할 수 있는 기능을 제공하는 병렬 루프로 변환된다. 만약 수백만 번의 반복 작업을 포함한 루프를 실행한다면 수백만 개의 쓰레드를 사용할 수 있는 기회가 제공되겠지만 실제로는 하드웨어와 효과적으로 매치될 수 있게 제한적으로 쓰레드를 만드는 것이 현실적이다. Listing 1과 2에서 OpenMP 내의 loop에서 하나의 directive나 pragma의 추가를 통해 쓰레드를 만드는 매우 간단한 예제를 볼 수 있다. 일반적으로 성능 최적화를 위해서는 약간의 추가적인 재구성이나 재배열 작업이 필요하다. 하지만, 그런 작업은 완료 후 다시 보면 매우 자연스러운 것으로 보이며 프로그램을 꼼꼼하게 살펴보면 directive가 주목할 코드 변경사항임을 알 수 있다.

```
!$OMP PARALLEL do PRIVATE(j,K)
do   i=1, M
        ! each thread will work i    ts own part of the problem
do   j=1, N
          do k=1, X
                ! calculations
end do
end do
end do
```

프로그램 예제 1. OpenMP directive를 사용한 많은 쓰레드 생성을 위한 Fortran do loop 변경

```
 #pragma omp parallel for private(j,k)
for   (i=0;   i<M; i++) {
        //   each thread will work its own part of the problem
for   ( j=0; j<N; j++) {
          for (k=0; k<X; k++)   {
            // calculations
          }
        }
  }
```

프로그램 예제 2. OpenMP pragma를 사용한 많은 쓰레드 생성을 위한 C for loop 변경

병렬 프로그램 성능 극대화 방안

애플리케이션을 Intel® Xeon® 프로세서에서 실행하든 Intel® Xeon Phi™ 에서 실행하든 간에 높은 성능을 얻기 위해서는 다음 2가지 기본적인 사항을 먼저 고려하여야 한다.

1. 스케일링: 애플리케이션의 스케일링이 고도의 병렬화 기능을 제공하는 Intel® Xeon Phi™ 코프로세서를 활용할 준비가 되었는가? 가장 확실한 증거는 일반적으로 Intel® Xeon® 프로세서상에서도 스케일링을 보여준다는 사실이다.
2. 벡터라이제이션과 메모리 사용량:
   A. 벡터 유닛을 빈번하게 사용하는가?
   B. Intel® Xeon® 프로세서에서 제공되는 것 보다 더 많은 로컬 메모리 대역폭을 활용할 수 있는가?

만약, 애플리케이션에서 이들 두 가지 기본사항이 모두 사실이라면 고도의 병렬화 및 뛰어난 전력효율을 제공하는 Intel® Xeon Phi™ 코프로세서는 충분히 검토 및 평가해 볼 가치가 있다.

성능 향상을 위한 프로그램 변환

Intel® Xeon Phi™ 코프로세서를 사용할 경우 별도의 변환작업을 하지 않아도 어느 정도의 성능향상을 기대할 수는 있으나 최고의 성능구현을 위해서는 변환작업을 하는 것이 좋다. 다음과 같은 사항을 고려하여야 최고의 성능을 얻을 수 있다.

- 메모리 액세스 및 loop 변환 (예: cache blocking, loop unrolling, prefetching tiling loop interchange, alignment, affinity)
- 벡터라이제이션은 unit-stride에 대해 최고의 효과를 냄. 데이터 구조 변환은 unit-stride (예: indirect access 대신 packed array를 사용하기 위한 AoS3 에서 SoA4 로의 변환 혹은 재코딩)와 함께 데이터 액세스 량을 증가시킬 수 있음.
- full-vector를 사용하는 것이 가장 좋음. 그리고 이를 위해 데이타 변환이 반드시 고려되어야 함.
- 벡터라이제이션은 잘 배열된 데이터와 함께 사용하여야 최고의 효과를 냄
- 큰 사이즈의 페이지를 고려 할 것(널리 사용되는 Linux libhugetlbfs 라이브러리 추천)
- 병렬화와 벡터라이제이션에 잘 맞는 알고리즘을 선택하거나 잘 맞도록 변경.

코프로세서의 주요 사용 모델: MPI 와 offload

기존 Intel® Xeon® 프로세서 기반의 호스트 시스템에서 프로그램을 어떻게 작성하여야 하는지는 이미 잘 알고 있을 것이다. Intel® Xeon PhiTM 을 활용하려면 어떻게 애플리케이션을 작성하여야 할까? 기본적으로 두가지 접근 방법이 있다.

1. 프로세서 중심의 " offload" 모델: 프로그램이 프로세서상에서 실행되는 것으로 보이며 선택적으로 작업을 코프로세서에 offload방식으로 전달 처리

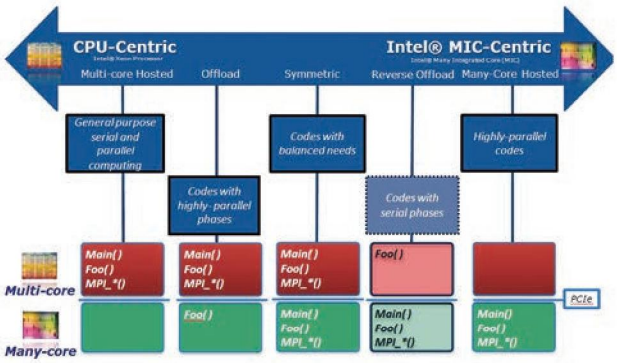


그림 15. 실행 모델 스펙트럼

2. "native" 모델: 프로그램이 프로세서와 코프로세서상에서 독자적으로 실행되며 여러가지 기법으로 서로 통신

그림 15에서 보듯이 애플리케이션의 구성방법에 따라 Intel MIC 아키텍쳐가 지원하는 다양한 실행 모델이 제공되고 있다. 실행모델은 크게 다음 두 가지에 의해 결정된다.

- 메인 프로그램이 시작되는 위치
- 애플리케이션 실행 시 순차처리와 병렬처리 부분

각 실행 모델의 의미는 다음과 같다.

Multicore-Hosted Model

일반적으로 알려진 표준 Intel® Xeon® 기반 클러스터 링이며 Intel® Xeon Phi™ 가 관여하지 않거나 필요하지 않다. 범용의 순차 및 병렬처리 컴퓨팅이 지원된다. 위의 그림 15의 예에서는 main()과 작업을 처리하는 foo() 가 멀티코어, 즉 Intel® Xeon® 프로세서 상에서 실행된다..

Many-Core Hosted Model

스펙트럼의 맨 끝부분에 있는데, 이것은 Intel® Xeon Phi™ 코프로세서를 위한 경우와 유사하며 고도로 병렬화된 코드에 적합하다. ACN (autonomous compute node) 머신 모델과 인텔의 소프트웨어 스택 및 개발 툴을 통해 구현 가능하다. 그림15의 예에서는 Main() 과 foo()가 전체적으로 many-core, 즉 Intel® Xeon Phi™ 코프로세서상에서 실행된다. Intel® Xeon Phi™ 의 각 코어는 MPI rank로 취급된다. 이 같은 실행 모델은 때때로 "native"모델이라 불린다.

Offload Models

offload 모델(forward 및 reverse)에서는 메인 프로그램이 Intel® Xeon® 이나 Intel® Xeon Phi™ 에서 각각 실행된다. 그러나 코드의 특정 부분은 원격으로 실행된다. forward offload의 경우 고도로 병렬화된 작업이 Intel® Xeon® 으로부터 Intel® Xeon Phi™ 로 전달된다. reverse offload의 경우는, 고도로 병렬화된 작업이 Intel® Xeon Phi™ 에서 Intel® Xeon ® 으로 전달된다. 연산을 위한 통신비율이 낮은 경우에는 offload 모델이 비효율적이다.

```
# define NSET 1000000
int main ( intargc, const char** argv )
{           longinti;
            floatnum_inside, Pi;
            num_inside = 0.0f;
            #pragma offload target (MIC)
    #pragma omp parallel for reduction(+:num_inside)
            for(i = 0; i< NSET; i++ )
            {           float x, y, distance_from_zero;
                        // Generate x, y random numbers in [0,1)
                        x = float(rand()) / float(RAND_MAX + 1);
                        y = float(rand()) / float(RAND_MAX + 1);
                        distance_from_zero = sqrt(x*x + y*y);
                        if ( distance_from_zero<= 1.0f )
                                    num_inside += 1.0f;
                }
    Pi = 4.0f * ( num_inside / NSET );
printf("Value of Pi = %f \n",Pi);
}
```

Symmetric Model

symmetric model은 메인 프로그램이 Intel® Xeon Phi™ 와 Intel® Xeon® 양쪽에서 실행되며 사용자가 균형을 조절하게 된다. MPI는 전형적으로 symmetric 모델을 사용하기 위한 방법을 제공한다.

(예제) Pi 계산 프로그램 – CPU버전에 #pragma offload target (MIC)한줄 만 추가

컴파일러 및 프로그래밍 모델

일반적으로 널리 사용되는 프로그래밍 언어 중에서 병렬화를 위해 특별히 설계된 것은 없다. 여러모로, Fortran이 병렬 프로그래밍을 지원하기 위해 DO CONCURRENT 같은 새로운 기능 추가에 가장 적극적이다. C 사용자는 OpenMP뿐만 아니라 Intel Cilk™ Plus를 사용할 수 있다. C++ 사용자는 Intel Threading Building Blocks와 최근에 Intel Cilk™ Plus 도 활용할 수 있게 되었다. C++ 사용자는 Open MP와 OpenCL 모두 사용할 수 있다.

Intel® Xeon Phi™ 코프로세서는 Intel® Xeon® 프로세서에서 사용 가능한 것과 동일한 개발 툴, 프로그래밍언어, 프로그래밍 모델 등을 사용할 수 있다. 그렇지만, 코프로세서가 본질적으로 고수준의 병렬처리를 위해 설계되었으므로 몇몇 프로그래밍 모델이 더욱 관심을 끌고 있다.

개발자들과 그 동안 잘 진행해온 사항들에 기반하여 몇 가지 권고사항이 있다. Fortran 프로그래머는 OpenMP, DO CONCURRENT그리고 MPI를 사용하는 것이 좋다. C++ 프로그래머는 IntelTBB, Intel Cilk™ Plus 및 Open MP를, C 프로그래머는 Open MP와 Intel Cilk™ Plus를 사용하는 것이 바람직하다. Intel TBB는 C++ 템플릿 라이브러리인데 뛰어난 작업기반 부하 밸런싱을 제공한다. Intel TBB가 명시적으로 벡터라이제이션을 지원하지는 않지만, 벡터라이제이션을 위한 솔루션 선택과는 상관이 없다. Intel TBB는 오픈 소스 기반이며 다양한 플랫폼과 대부분의 운영체제 및 프로세서를 지원한다. Intel Cilk™ Plus는 태스킹과 벡터라이제이션 양쪽 모두를 지원한다는 점에서 약간 더 복잡하다. Intel Cilk™ Plus는 Intel TBB 와 상호운용 성을 완벽히 제공한다.

Intel Cilk™ Plus는 Intel TBB 보다 간결한 작업처리 기능을 제공하지만 최적화를 위해 프로그래밍 언어의 키워드를 사용함으로써 완벽하게 컴파일러를 지원한다. 또한 Intel Cilk™ Plus는 벡터라이제이션을 돕기 위한 기본 함수들과 배열(array)구문 및 "#pragma SIMD" 를 제공한다. 배열구문을 가장 잘 활용하는 방법은 캐시를 위한 blocking을 함께 사용하는 것이다. 그런데 이것은 A[:] = B[:] + C[:]같은 형식으로 구성하는 native방식의 사용법을 의미하는데, 안타깝게도 대규모 배열에서는 성능이 저하될 수 있다. 배열구문을 최적으로 사용하는 방법은 하나의 문장에 사용되는 벡터 길이를 native 벡터길이의 1배 정도로 짧게 만드는 것이다. 결국, 프로그래머에게 있어서 가장 중요한 것

은 Intel Cilk™ Plus가 "#pragma SIMD"로 불리는 컴파일러를 위한 필수 벡터라이제이션 pragma를 현재 제공한다는 사실이다. OpenMP가 병렬화를 위해 사용되었던 것처럼 "#pragma SIMD"는 벡터라이제이션을 위한 것이다. Intel Cilk™ Plus는 컴파일러의 지원을 필요로 한다. 현재 Windows, Linux 및 Apple OS X환경에서 지원된다. 또한 gcc에서도 사용 가능하다.

만약 OpenMP나 MPI와 이미 친숙하다면 Intel® Xeon Phi™ 코프로세서를 잘 활용할 준비가 되어 있다는 것을 의미한다. 추가 옵션들은 사용을 거듭할 수록 매력적인 것이 될 것이다. 그렇지만 OpenMP와 MPI만으로도 효과적인 벡터라이제이션을 사용한다면 충분히 좋은 결과를 얻을 수 있다. 특히 정렬(alignment) 및 unit-stride 액세스 같이 효과적인 벡터라이제이션을 위해 적용 가능한 추가 고려사항과 함께 Fortran으로 코드를 작성한다면 자동-벡터라이제이션만으로도 충분할 것이다. Intel Cilk™ Plus(Fortran에서도 사용가능)의 "#pragma SIMD" 기능은 충분히 주목할 가치가 있다. 아마도 조만간 어느 순간엔가 이미 OpenMP의 일부가 되어 있는 것을 보게 될지도 모른다.

MPI는 완벽한 유연성과 뛰어난 기능으로 이미 수 십 년 동안 많은 프로그래머들에 의해 사용되어 왔다. 최근에는 공유 메모리 프로그래머들이 내장형 작업처리 모델과 함께 Intel TBB 와 Intel Cilk™ Plus를 사용하고 있다. Intel TBB는 C++ 커뮤니티에서 폭넓게 사용되고 있으며, Intel Cilk™ Plus는 C프로그래머에게 C 와 C++ 프로그램 벡터라이제이션에 도움을 주기 위해 Intel TBB의 기능을 확장해 준다.

캐시 최적화

캐시를 가장 효과적으로 사용하는 방법은 참조(reference)의 집약성(locality)를 극대화하고, L2캐시에 맞게 봉쇄(blocking) 하는데 집중하는 것이다. 그리고, 프리페칭(prefetching)이 활용되고 있는지(하드웨어에 의해, 컴파일러에 의해, 라이브러리에 의해 혹은 명시적인 프로그램 제어에 의해) 확실하게 하는 것이 좋다.

데이터 집약성(locality)을 512K 로 맞추거나 혹은 코어당 L2 캐시 사용량을 적게 하면 일반적으로 전체 L2캐시 사용률이 최적인 상황이 된다. 코어당 4개의 모든 하드웨어 쓰레드가 "per core" L2 캐시를 공유한다. 그렇지만 캐시에 대한 고속의 액세스는 다른 코어들과 연관되어 있다. 특정 코어에 의해 사용되는 데이터는 로컬 L2 캐시(동일 칩상의 복수개의 L2 캐시일 수도 있음)의 공간을 점유하게 된다. Intel® Xeon® 프로세서의 경우 "cross-socket" 형태의 공유로 인한 페널티가 생기는데, 약 16개의 쓰레드(8코어의 경우 코어당 각각 2개의 쓰레드) 이후부터 발생한다. Intel® Xeon Phi™ 코프로세서는 200개 이상의 쓰레드에서도 더 적은 페널티를 가진다. 먼저 어떤 하나의 코어에서 사용되는 쓰레드(최대 4개 까지)에 적용되는 집약성을 갖게 하고, 그런 다음 모든 코프로세스에 있는 쓰레드 전체에 대해 집약성을 맞추는 것이 효과적이다.

기본 값과 자동 처리가 상당히 좋을 수도 있는 반면, 집약성을 위해 설계된 코드가 최적으로 실행되게 하는 방법은 OpenMP환경에서 KMP_AFFINITY와 MPI환경에서 I_MPI_PIN_DOMAIN을 사용하는 것처럼 affinity 관련 세부사항을 프로그램에 포함시키는 것이다. 동일 코어상에서 쓰레드를 공유하는 것이 큰 이점이 있는 반면, 코프로세서상

에서 어떤 코어가 다른 코어와 얼마나 가까이 있는지를 기준으로 성능 변화를 기대하면 안 된다는 것에 주목해야 한다. 이러한 사실이 놀랍겠지만, 이런 측면에서 하드웨어 설계가 너무 뛰어나서 코프로세서의 코어간 근접성으로 인한 성능의 향상은 찾아 볼 수가 없다. 따라서 코어에 대해 집약성 이상으로 배치를 최적화하고 그런 다음 코어 전반에 걸쳐 로드 밸런싱하는데 시간을 쏟아 붇는 것은 (예를 들면 KMP_AFFINITY =scatter를 사용하여 라운드로빈 방식의 작업을 할당하는 것) 별로 권장할 만한 것이 아니다.

코프로세서는 어떤 페이지 내에서 최초의 캐시 미스에 의해 시작되는 L2로의 하드웨어 프리페칭 기능을 가지고 있다. 인텔 컴파일러는 기본적으로 루프내의 메모리 참조를 위해 소프트웨어 프리페치 기능을 사용한다. 전형적으로 컴파일러는 메모리 참소당 두 개의 프리페치를 제기하는데, 하나는 메모리에서 L2로, 다른 하나는 L2에서 L1으로 프리페치 하기 위한 것이다. 프리페치의 거리는 루프내의 작업 처리량에 기반하여 컴파일러에 의해 계산된다. 명시적 프리페칭은 프리페치 pragma (C/C++에서의 #pragma prefetch, 혹은 Fortran에서의 CDEC$ 프리페치) 혹은 프리페치 intrinsic (C에서의 _mm_prefetch, 혹은 Fortran에서의 mm_prefetch)과 함께 추가될 수 있다.

프리페치 intrinsic을 수작업으로 하려면 컴파일러의 프리페칭을 명시적으로 끌 수도 있다(모든 컴파일러 프리페치를 끄려면 –opt–prefetch=0 혹은 컴파일러 프리페치를 L2내부로 하려면 -opt–prefetch–distance=0,2).

소프트웨어 프리페치는 성능 카운터에 의해 캐시 미스로 카운트되지 않는다. 이 말은 성공적으로 모든 데이터 스트림을 프리페치를 통해 가져올 수 있을 때, 루프 내에서 “캐시 미스”가 본질적으로 0 (메모리 액세스 시 캐시 미스 횟수가 낮은 한자리수의 백분율을 기록하는 것을 목표로)이 되도록 연구 할 수 있다는 것을 의미한다.

프리페칭은 메모리에서 L2로, 또 별도로 L2에서 L1사이에 필요하다. 메모리에서 L1으로 직접 프리페치를 활용하는 것으로 대폭적인 성능 향상을 기대할 수는 없다. 왜냐하면 그 과정에서 발생하는 지연현상(latency)이 하드웨어에서 제공하는 L1 프리페치보다 훨씬 더 많이 발생하기 때문이다. 이러한 제약사항은 쓰레드당 데이터 스트림 수가 8개 보다 적고 프리페칭이 각 데이터 스트림에 대해 활발하게 사용될 때 데이터 스트림을 구성하는 것이 최선임을 의미한다. 경험상, 코어당 활성화된 프리페치의 수가 30 내지 40개 정도로 관리되어야 하거나 활성화된 데이터 스트림 전반에 걸쳐 분할 되어 있어야 한다.

캐시 이외에도, TLB내에서의 핫스팟(hot spot)을 피하기 위해 크고 작은 페이지를 사용하거나, 데이터 스트림을 구성하고, 또 데이터를 구성하도록 할 수 있는 기능을 포함하여 TLB의 추가 튜닝을 위해 특정 메모리 변환 작업을 적용할 수도 있다.

맺음말

성공적인 병렬 프로그래밍이란 “선호하는 프로그래밍 언어와 병렬 모델을 기반으로 벡터를 사용하는 많은 쓰레드를 가진 프로그램”이라 정의할 수 있다. 대부분의 애플리케이션이 아직은 프로세서에서 지원하는 최고 수준의

병렬화가 되어 있지 않기 때문에 더 많은 병렬화를 위해 어떻게 코드를 재구성 하여야 하는가가 프로세서와 코프로세서로 부터 최고의 성능을 끌어내는데 꼭 필요한 부분이다. Intel® Xeon® 과 Intel® Xeon Phi™ 를 함께 사용하는 환경에서의 이러한 재구성 작업은 범용 프로그래밍 언어와 모델, 그리고 개발 툴들을 사용할 수 있기 때문에 프로세서와 코프로세서 모두에서 큰 효과를 기대할 수 있다는 장점이 있다.