Research Challenges in Many-core SoC Designs

정의영 | 유승주
연세대학교, 포항공과대학교

요 약

본고에서는 최근 학계에서뿐만 아니라 Intel, nVidia 등의 반도체 설계업계에서도 차세대 system-on-chip (SoC) 구조로 제안하고, 실제품 설계까지 진행 중인 many-core SoC의 research challenges를 알아본다. 이러한 challenges는 architecture, software, application의 3가지 면에서 살펴 보는데, 각 분야에서 주요 문제들을 고찰하고, 이 문제들을 해결하기 위해 현재 진행 중인 주요 연구 방향들을 살펴보고자 한다.

I. 서 론

본고에서는 system-on-chip (SoC) 설계의 새로운 방향인 many-core SoC의 기술적인 그리고 실용적 issue들과 이를 해결하려는 새로운 research idea들에 대해 알아본다. 2004년 Intel의 dual core Pentium 출시는 Many-core SoC 시대를 예상할 수 있게 하는 첫 사건이라고 볼 수 있다. 이와 비슷한 시기의 Cisco에서 구현한 network processor chip은 이미 188개의 작은 core들을 하나의 silicon die에 구현하였고 [1], Intel은 2007년 80개의 core를 집적한 Polaris를 research prototype으로 발표하였고 [2], nVidia는 100여개의 processor 를 집적한 Tesla T10P를 GPU 시장에 출시하였다 [3]. 그리고, 최근 Intel은 최대 64개의 core와 분산 L2 cache 구조로 이루어진 Larrabee를 GPU향으로 출시 준비 중이다 [4]. Many-core 는 위와 같이 그 필요성을 절감하는 industry 에서 아주 적극적으로 적용하고 있다고 할 수 있겠다.

이와 같은 many-core SoC가 필요하게 된 배경은 ILP (instruction level parallelism) wall과 power wall의 크게 2가지로 볼 수 있다. 2002년 이전까지 Pentium과 같은 processor의 성능은 크게, transistor의 성능향상과 processor 구조의 개선이라는 2가지 방법에 의해 매년 50% 이상 지속적으로 향상되었다. Processor 구조의 경우 초기의 in-order 구조를 (issue width를 늘리며) out-of-order로 하여 하나의 cycle에 수행 가능한 instruction 수, 즉, IPC (instructions per cycle)을 늘렸으며, branch prediction과 같은 speculative 수행을 통해, program의 control flow에 의한 제약을 극복하여 더욱 더 IPC를 향상시켰다. 또한, cache miss penalty를 최소화하기 위해, multi-level cache, victim cache등의 다양한 구조를 개발 적용해 processor의 성능향상 trend를 유지해 왔다. ILP wall은 이러한 하나의 processor 내부 구조 향상을 통한 성능 향상이 한계에 이르렀다는 것을 의미한다.

ILP wall은 크게 2가지 면에서 비롯된다. 하나는 processor 성능향상을 위해 지불하는 비용(area/power)이 지나치게 커져, 성능개선 대 비용의 비율이 아주 낮아졌다는 점이다. 이를 단적으로 보여주는 표현이 Pollack의 rule이다. 이는 일반적으로 processor에 들이는 비용을 2배로 강화해도 성능향상은 100%가 아닌 40% 수준 밖에 얻을 수 없다는 사실이다. 이렇게 비용이 증가하게 된 데는 processor 에서 수행하는 프로그램 자체가 가지는 instruction 수준의 병렬성이 제한적이라는 것이 두 번째 이유이다.

하나의 processor 구조를 개선하는 것이 벽에 부딪혔는데,

이에 대한 자연스런 대안이 processor의 수를 늘리는 것, 즉 many-core 설계인데, 이를 통해 구현 가능한 계산량의 증가라는 기존 경향을 유지할 수 있게 되는 것이다.

두번째 문제인 power wall은 위의 성능개선/비용의 계산에서 비용에 속하는 전력소모가 문제가 된다는 것인데, 전력소모 문제는 processor 구조와 트랜지스터 특성이라는 두 가지 면에서 이해할 수 있다. Processor 구조 면에서는 성능 향상을 위해 적용한 구조적 해법들, 특히, out-of-order 및 speculative 수행 방법은 하나의 instruction 수행을 위해 간단한 구조보다 더 많은 동작, 즉, 전력소모를 필요로 한다. 예를 들어, branch prediction의 경우, speculation이 실패했을 때는 speculative 하게 수행한 instruction들의 결과가 필요 없게 된다. 즉, 이 instruction들 수행에 소모한 전력은 고스란히 비용으로 추가되어 전체 processor의 전력소모를 증가시키게 된다. 이러한 전력소모 증가 문제를 해결하는 좋은 방법이 병렬수행이다. 이는 전자회로의 전력소모 및 트랜지스터의 동작가능 주파수와 전원전압과의 관련 때문이다. 우선, 전자회로의 전력소모는 power ~ CV^2 이다. (여기서, C는 capacitance, V는 전원전압). 트랜지스터의 최대 동작주파수 f와 전원전압은 비례관계를 갖는다. 따라서, 동일한 일을 하나의 processor에서 V의 전원전압으로 1초 동안 수행하는 것 (에너지소모 = $f * C * V^2$) 보다, 2개의 processor에서 일을 나눠서 각각이 V/2의 전원전압으로 수행하는 것 (에너지소모 = $f/2 * C * V^2/4 + f/2 * C * V^2/4$)이 전체 에너지 소모를 1/4로 줄일 수 있게 된다.

위와 같은 ILP 및 power wall은 기존의 multi-core SoC의 필요성과 궤를 같이 한다. 여기에 더해 many-core SoC 만의 필요성은 SoC상에 구현할 새로운 application에 의한 것이라 볼 수 있다. 이미 시장에 나와 있는 many-core SoC 및 core 수가 많은 multi-core SoC를 보면 network processor, GPU, server향 칩들인데 이들은 모두 대량의 병렬 (massively parallel) application을 수행하는 것들이다. 예를 들어, server의 경우 아주 많은, 그러나 서로 독립적인 transaction들, 즉 대량의 병렬 계산을 수행한다. GPU에서 수행하는 pixel processing 역시 대량의 병렬 계산이다. 이러한 대량의 병렬 계산은 위와 같은 application 분야에 머무르지 않고, 기존의 multimedia, mobile을 포함하여, 의공학, 금융에 이르기까지 소위 RMS (II.4에서 설명)로 불리는 보다 광범위한 적용 범위를 가질 것으로 예상된다.

본고에서는 이러한 many-core SoC 설계를 위해 새로이 풀어야 하는 문제를 architecture, software, application의 3가지 면에서 살펴보고자 한다.

II. 본 론

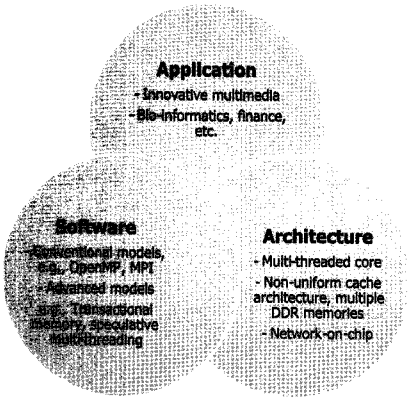
II.1 Overview of many-core SoC research challenges

(그림 1)은 architecture, software, application 각 분야에서 issue들을 정리한 것이다. Architecture는 many-core 구성에서 '전체를 이루는 부분' 의 기능을 정하는 문제 (즉, 하나의 core가 가져야 하는 기능 및 구조를 정하는 문제), 대량의 병렬성을 지원하기 위한 memory hierarchy 문제, 그리고 core들 간의 그리고 core와 memory 간의 연결 문제로 나누어 볼 수 있다. 소프트웨어의 경우 '병렬화를 어떻게 할 것인가' 가 주된 issue인데, 이를 위해 다양한 병렬 programming 방법들이 many-core architecture와 application의 context 상에서 효과적으로 적용되어야 한다.

Application 에서는 기존에는 수퍼컴퓨터에서나 수행이 가능했던 기능을 이제는 계산 기능이 획기적으로 증가한 하나의 칩 상에서 수행하는 것을 가정해 보자. 우선은 기존 SoC에서 계산량의 주요 부분을 차지한 멀티미디어 기능(예, 3D graphics)의 고급화를 생각해 볼수 있다. 이에 더 나아가 의공학, 금융과 같은 대량의 데이터를 복잡한 모델을 이용 분석하고, 원하는 결과를 찾아내고, 생성해 내는 새로운 분야가 many-core SoC 적용이 가능한 분야가 될 것인데, 이러한 분야로의 응용 가능성을 파악하고, 준비하는 것이 중요한 issue 일 것이다.

II.2 Architecture

Many-core의 기본 구성요소인 하나의 core가 가지는 성능/비용의 비는 아주 중요하다. 예를 들어, 아주 단순한 core를 사용할 경우 비록 하나의 chip에 구현 가능한 core의 수는 아주 많아지지만, 수행 가능한 계산량은 크지 않을 수 있다. Core의 IPC (instruction per cycle)이 낮은 경우 전체성능



(그림 1) Overview of many-core SoC research challenges

/비용(칩면적)은 좋지 않게 된다. 즉, 성능만을 놓고 본다면 주어진 칩 면적에 (정확하게는 core에 할당된 칩 면적에) 각 core의 IPC * core의 개수를 최대로 하는 core를 선택하는 것이 필요하다 [5].

Core의 IPC는 core에서 수행되는 application에 따라 차이가 크기 때문에 이러한 core의 선택에 있어서 application을 구체적으로 고려하지 않는 (전체 가능한 application들에 대해 평균 성능을 최대화하는 core를 선택하는 것과 같은) 일률적인 방식은 현실적이지 못할 것이기 때문에, 대표적인 규칙 (rule of thumb)은 얘기하기 어려울 것이다. 그러나, 최근의 core 구현 동향을 보면, 앞으로 many-core SoC에서 사용될 core의 구조에 대한 예상은 어느 정도 가능할 수 있다. 그 단서로 최근 Intel에서 발표한 ATOM과 Larrabee processor, Sun의 Rock processor, nVidia Tesla, IBM Cell processor를 들 수 있다. 우선, ATOM, Larrabee, Rock의 공통점은 각 core가 lightly multi-threaded 구조를 가진다는 점이다. 즉, 2~4개 정도의 thread를 지원하는 SMT (simultaneous multi-threading) core이다. 따라서, thread 수준의 병렬성 (thread level parallelism, TLP)을 가지는 application 수행 시 효과적이다. 그리고, Tesla와 Cell의 경우는 각 core가 SIMD (single instruction multiple data) 구조를 갖는다. 이는 ILP가 큰 병렬 연산의 수행에 잇점을 갖는다. 위와 같이, 크게 TLP를 위한 lightly SMT core들과 ILP를 위한 SIMD core의 2가지 종류(물론, 이들 간의 사용비율은 적용 제품군에 맞게 정해야 한다)로 many-core SoC는 구성될 가능성이 높다고 볼 수 있다.

Many-core SoC는 주어진 chip 면적 및 전력소모 조건을 만족하면서 처리 가능한 계산량을 극대화하는 것이 설계의 목표일 것이다. 계산량이 늘어나면 그에 비례에 늘어가는 것이 필요한 memory bandwidth 이다. Many-core의 memory hierarchy 문제는 예를 들면 다음과 같은 것이 될 것이다. "기존의 4 core (L1 cache는 core 내부에 있다고 가정) 에서 필요한 memory bandwidth를 맞추기 위해 하나의 L2 cache, external DDR memory가 필요했다면, 100개의 core를 위해서는 25개의 L2 cache, 25 개의 DDR memory가 필요할 것인가?". 이 경우 25개의 DDR memory는 쉽게 현실적인 답이 되기 어렵다는 것을 알 수 있다. 또한, wire delay 문제로 25개의 cache를 1개 cache의 경우와 동일한 성능으로 사용하는 것은 현실적으로 아주 어려울 것이다 (물론, transmission line을 사용해 동일한 접근시간을 보장하려는 접근 방법이 있지만 [6], 이 경우 아주 늘어나는 면적비용과 구현가능성의 문제를 해결해야 한다). 그러나, 이 경우 100개 (또는 그 이상의) core가 필요로 하는 memory bandwidth를 효과적으로 (즉, memory hierarchy에 사용 가능한, 이미 주어진 비용 범위 내에서) 제공하지 못하면, 이러한 many-core 구현은 어려울 것이다.

이러한 memory bandwidth 문제를 해결하기 위한 방향은 현재까지 L2 cache와 DDR memory 에 대해 2가지 큰 흐름이 있다고 하겠다. 칩 상의 L2 (그리고 필요하다면 L3) cache의 성능의 경우, wire delay로 인해, 전체 L2 cache를 synchronously 동작시킬 경우 cache 접근 지연시간이 가장 멀리 있는 memory bank의 접근시간에 의해 결정되어, 아주 낮은 성능을 얻게 되는 것이 가장 큰 문제이다. 이를 해결하기 위해, NUCA (non-uniform cache architecture)의 적용은 필수적이라 하겠다 [7]. NUCA는 하나의 chip 상에서 특정 core와 지리적으로 가까이 있는 L2 cache memory bank와 멀리 있는 것 간에는 접근 시간 (access latency)이 다른 것을 그대로 인정하고 받아들이는 구조이다. 즉, L2 cache 접근 시간이 data가 있는 위치(L2 cache memory bank의 위치)에 따라 달라진다. NUCA는 L2 cache memory bank 상에서의 data promotion/demotion, bank sharing등의 새로운 해결할 문제를 제시하는데 최근 이에 대한 다양한 해법에 대한 연구가 활발히 진행되고 있다 [7].

NUCA 를 적용하여 효과적인 cache 구조를 가지는 경우에

도 하나의 many-core silicon die에서 필요로 하는 external memory에 대한 bandwidth는 아주 높을 것이다. 이를 위해 기존의 DDR memory와의 연결방식인 die의 주변에 있는 pad를 이용하는 경우, 사용 가능한 pad 수의 제약으로, memory 수를 늘리는 방식의 memory bandwidth 향상은 어렵게 된다. 대신, memory I/O의 동작 주파수를 늘리는 방법도 고려할 수 있는데, 이 경우는 동작 주파수에 비례해서 늘어나는 전력소모가 큰 문제가 된다 [5]. 이러한 external memory bandwidth 문제를 해결하기 위한 현실적인 해법으로 최근 3D die stacking을 통한 memory die와 SoC die간의 수직적 연결 (through silicon via, TSV 또는 face-to-face)을 적용하는 것이 활발히 연구되고 있다 [5]. 즉, SoC die 내부의 여러 지점에 memory die와 연결되는 via를 만들어 memory die와 연결시키는 것이다. 이러한 연결은 연결자체가 가지는 면적비용과 test 등의 issue가 해결될 경우, memory bandwidth를 향상시키는 점에서는 큰 잇점을 갖는다. 즉, 기존의 주변 pad를 이용하는 방식 대비 획기적으로 연결의 수를 늘릴 수 있다. 이는 2차원 평면상에 TSV 등을 위치시키기 때문에 die 주변에만 pad를 위치시키는 경우보다 훨씬 많은 수의 연결이 구현가능하기 때문이다.

100여개에 이르는 core들은 core들 간에, 그리고 core와 memory 간의 통신을 필요로 하는데, 이러한 통신에서는 많은 수의 traffic이 병렬적으로 전달되며, 전체 통신 대역폭의 합이 아주 크게 된다. 칩 상의 통신을 구현하는데 사용한 기존 shared bus나 crossbar 구조로는 이러한 병렬적이고 높은 대역폭의 통신을 지원하기 어려워, 2000년대 초부터 network-on-chip (NoC)이 활발히 연구되어 왔고, 최근 Intel에서는 이를 research prototype (Polaris)과 GPU product (Larrabee)에 적용하기에 이르렀다. 이러한 NoC는 router들이 대개 짧은 link들로 router 간에, 그리고 processor 또는 L2 memory bank와 연결된다. 짧은 link는 고속 동작을 가능해 link 대역폭을 쉽게 높일 수 있고, total link의 수 및 bi-section bandwidth가 크도록 NoC topology를 설계하여 칩 상의 통신에 필요한 전체 통신 대역폭 조건을 만족시킬 수 있게 된다.

NoC는 70~80년대에 기존 computer interconnection network 분야에서 연구된 내용을 칩상의 구현이라는 면에서 특화한 방향으로 연구가 주로 진행되어 왔는데, 기존 대비 NoC가 가지는 대표적인 차이점은 latency의 중요성과 memory에 대한 고려라고 볼 수 있다. NoC의 latency는 processor의 L2 cache miss penalty를 결정하므로 최소화가 필수적이다. 또한, memory는 NoC를 통한 통신의 대부분이 memory와 core (또는 cache) 간의 통신이기 때문에 memory를 고려한 NoC 설계는 필수적이다. 이들 각각을 위해 최근 express virtual channel [8], transmission line 기반 hybrid NoC [9], multiple DDR memory를 효과적으로 접근하기 위한 request 병렬화 구조 [10] 등등이 활발히 연구되고 있다.

II.3 Software

Many-core SoC 설계에 있어 기존 SoC 설계 대비 가장 큰 차이점은 병렬 프로그래밍의 필요성일 것이다. 많은 수의 core의 사용률을 높이기 위해 application 내부에 있는 병렬성을 최대로 찾아내어 병렬 프로그래밍을 하는 것이 필수적이다. 이러한 병렬 프로그래밍 방법은 shared memory, message passing, streaming의 3가지 방법을 application과 SoC architecture에 맞게 적용한다. 즉, symmetric multi-processor (SMP)와 같이 동일한 core들이 같은 coherent cache 구조를 가지고 있는 경우(예, Larrabee)에는 OpenMP와 같은 shared memory model을, 공통된 coherent cache를 가지지 않은, 그러나 각각의 core의 local memory가 큰 경우 (예, IBM Cell)에는 message passing이나 streaming 프로그램을 주로 적용한다.

기존 processor 구조의 성능 향상의 많은 부분들이 speculative 수행에서 비롯된 것과 같이, 이러한 병렬 프로그래밍 분야의 연구 역시 현재 speculative 수행을 효과적으로 수행하는 방향의 연구가 활발히 진행되고 있는데, transactional memory와 speculative multi-threading (SpMT)이 대표적인 연구분야이다. Transactional memory는 병렬 프로그램을 구성하는 thread 간의 동기화에 기존 thread 프로그램에서는 lock을 사용하는데, 이 경우 lock을 얻기 위해 thread 수행이 지연되어, 병렬수행의 효과가 떨어지는 문제를 해결하기 위한 것이다. Transactional memory는 lock을 speculatively 관리하는 방법으로 현재 Sun, Microsoft 등에서 many-core SoC를 장착할 server 및 PC 향 프로그램 기술로 활발히 연구하고 있다 [11].

Application의 동작 자체가 병렬화하기 어려운 경우는 전통적인 병렬화로는 many-core 구조가 가지고 있는 병렬성을 충분히 활용할 수가 없다. SpMT는 순차적인 프로그램에서 나중에 수행되지만 수행확률이 높은 부분을 다른 core를 통해 미리 수행해서 전체 성능을 향상시키는 방법이다. Processor 구조에서 branch prediction을 프로그램 수준으로 높인 것으로 이해해 볼 수 있다. 이러한 기술은 many-core SoC 상에 기존 legacy code(대부분의 기존 코드들은 순차적으로 수행하도록 설계되어 있으므로)를 수행할 때 특히 유용한데, 이러한 SpMT는 compiler solution을 필요로 하므로 최근 SpMT를 위한 compiler에 대한 연구가 활발히 진행되고 있다 [12].

II.4 Many-Core SoC의 응용 분야

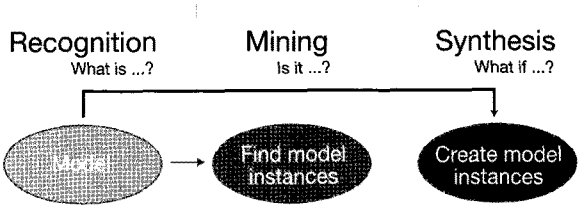
일반적으로 프로세서의 개수가 8개 이하의 병렬 구조를 다중 코어 구조라 하며, 그 이상의 프로세서를 채용하는 구조를 many-core 구조라 한다 [13]. 따라서, many-core구조는 다중 코어 구조에 비해 훨씬 강력한 계산 능력을 제공하므로 이에 걸맞는 응용 분야의 발굴이 필요한 실정이다. 그러나, 아직까지 이에 적합한 응용 분야의 발굴은 초기 단계에 머무르고 있다. 범용 컴퓨터 분야에서 many-core구조의 선도적인 역할을 하고 있는 인텔 (Intel)사는 앞으로 펼쳐질 many-core 구조의 응용 분야를 세가지로 분류하고 있다. 즉, 인지 (Recognition), 마이닝 (Mining), 합성 (Synthesis)의 세 분야로 요약하고 있으며, 이들의 첫 글자를 이용하여 RMS라고 한다[14]. 참고 문헌 [14]에서 정의한 위의 세가지 분야는 다음과 같이 요약될 수 있다.

인지는 컴퓨터를 이용하여 데이터 및 이미지를 검사하고, 컴퓨터가 식별한 특성을 바탕으로 수학적인 모델을 구축할 수 있는 machine-learning 능력을 의미한다. 한가지 예로서는 특정인의 안면에 대한 모델을 들 수 있다.

마이닝은 관심 대상의 패턴 또는 모델에 관련된 방대한 양의 실존 데이터를 분류하는 능력을 의미한다. 즉, 방대한 양의 데이터 중에 특정 모델의 인스턴스 (instance)를 찾아낼 수 있는 능력이다. 한가지 예로 다양한 해상도 (resolution) 및 광도 등을 반영한 다량의 이미지에서 특정인의 안면을 찾아내는 작업이 될 수 있다.

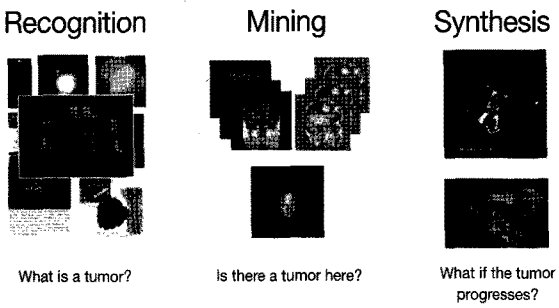
합성은 모델의 새로운 인스턴스를 구성하여 이론적인 탐색을 할 수 있는 능력을 의미한다. 특정인의 현재 이미지로부터 이 사람의 유년 혹은 노년의 모습이 어떨지를 예측하는 작업이 한가지 예가 될 수 있다.

이 세가지 분야는 각각 독립적인 특성을 가짐과 동시에 연관성도 내포하고 있으며, (그림 2)에 나타낸 바와 같다.



(그림 2) 인지, 마이닝, 합성간의 관계 [14]

(그림 2)에서와 같이 인지에서는 관심대상에 대한 모델을 구축하며, 이 모델을 바탕으로 마이닝에서는 방대한 데이터로부터 모델의 특성을 만족시키는 데이터 셋을 찾아낸다. 또한 합성에서는 존재하는 데이터로부터 찾아내는 개념보다는 주어진 모델 특성을 만족시키는 객체를 새로이 생성시키는 것이다. 이와 같이 RMS가 하나로 통합된 응용 분야의 좋은 예는 의공학 분야에서 찾을 수 있다. (그림 3)은 종양과 관련된 RMS 응용 분야를 보여주고 있다.

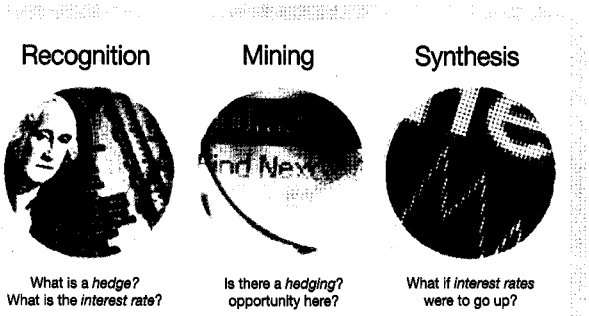


(그림 3) 종양 관련 RMS 응용 분야 [15]

(그림 3)의 첫 번째 그림은 컴퓨터에서 처리할 수 있도록 종양을 모델링하는 단계로 인지 단계에 해당한다. 두 번째 그림은 이 모델을 이용하여 인체 내에 종양이 존재하는지를 마이닝하는 단계에 해당하며, 마지막 그림은 이 종양이 더 진행될 경우를 예측해 보는 합성 단계에 해당한다. 의공학

분야는 이와 유사한 형태의 RMS 응용 들이 존재하므로, many-core 구조의 유망한 적용 분야가 될 것이다.

또 다른 RMS 응용 분야의 하나로 투자 (investment)를 생각해 볼 수 있다. 이미 금융 공학 등의 이름으로 금융 분야와 공학의 결합이 이루어 지고 있는 상황이며, 갈수록 증가되는 다양한 파라미터와 정보를 효율적으로 처리하기 위해서는 강력한 계산능력을 제공하는 컴퓨터가 필요한 실정이기 때문이다. (그림 4)가 이러한 예를 잘 나타내고 있다.



(그림 4) 금융 투자 관련 RMS 응용 분야 [15]

(그림 4)는 헤지펀드 (hedge fund)에 투자하는 예를 보여주는 것으로 헤지가 무엇인지, 또한 밀접한 파라미터인 이율 (interest rate)가 무엇인지 등을 인지 단계에서 모델링 한다. 마이닝 단계에서는 이러한 모델을 이용하여 여러 헤지펀드의 전략을 분석하여 적절한 투자처를 선택할 수 있다. 합성 단계에서는 이율 또는 여러 파라미터들이 변할 때 특정 헤지 펀드의 투자에 어떤 변화가 일어날 것인가 등을 예측하는 작업을 수행할 수 있다. 이러한 연구는 이미 IT분야의 연구자들에 의해서도 활발히 진행되고 있는 실정이다 [16].

이상에서 언급된 RMS의 응용 분야 예 이외에도 게임, 사업 (business), 가정 (home)등의 영역에서 응용 분야가 창출되리라 기대된다.

이와 같이 many-core 구조는 기존의 전통적인 IT 산업 이외에 다양한 응용 분야를 목표로 하고 있으며, 따라서 기존 IT 분야 보다 더 폭넓은 수요를 확보할 수 있다는 기대감을 주고 있다. 그러나 수요 증가의 속도는 빠르게 증가되고 있지 않기 때문에, 보다 장기적인 관점에서 접근해야 할 것이다. 이러한 저속의 수요 증가는 many-core구조의 응용 분야가 일종의 융합 분야라고 볼 수 있으며, 다양한 전문 지식의 결합이 생각보다 쉽지 않기 때문이다. 현재 의공학 분야는 IT와의 접목이 타 분야에 비해 빠른 속도로 진행되고 있으므로, 의공학을 포함한 바이오 응용 분야가 many-core구조의 가장 빠른 적용 분야가 될 것으로 기대된다.

III. 결 론

본고에서는 many-core SoC의 최근 연구 동향을 정리해 보았다. ILP 및 power wall로 시작된 many-core SoC는 architecture, software 및 application 면에서 새로운 연구개발 문제와 가능성을 제시한다. Architecture에서는 many-core에 적절한 core 선택, core들을 위한 memory hierarchy의 선택, core 및 memory간의 연결 네트워크라는 새로운 문제를 주며, software 분야에서는 병렬 프로그래밍의 적용과 성능 최적화 문제를 제시한다. Many-core SoC의 실용성을 결정할 application은 기존 멀티미디어 기반 응용의 고급화에서부터 의공학, 금융 분야에 이르기까지 아주 많은 가능성을 제공한다. 이러한 가능성을 실현하기 위해서는, architecture, software 분야의 기술적 문제 해결과 함께, application에서 요구하는 서로 아주 상이한 분야간의 융합이 필수적일 것이다.

참 고 문 헌

[1] Cisco CRS-1 Overview, available at www.cs.ucsd.edu/~varghese/crs1.ppt

[2] S. Vangal, *et al.*, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", IEEE Journal of Solid State Circuits, Jan. 2008.

[3] nVidia Tesla S1070, http://www.nvidia.com/object/tesla_s1070.html

[4] L. Seiler, *et al.*, "Larrabee: A Many-Core x86 Architecture for Visual Computing", ACM Transactions

on Graphics, August 2008.

[5] S. Borkar, "Thousand Core Chips - A Technology Perspective", Proc. Design Automation Conference, June 2007.

[6] B. Beckmann and D. Wood, "TLC: Transmission Line Caches", Proc. IEEE/ACM International Symposium on Microarchitecture, 2003.

[7] C. Kim, D. Burger, and S. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches", Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 2003.

[8] A. Kumar, *et al.*, "Express virtual channels: towards the ideal interconnection fabric", Proc. International Symposium on Computer Architecture, 2007.

[9] T. Krishna, *et al.*, "NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication", Symposium on High Performance Interconnects, Aug. 2008.

[10] W. Kwon, *et al.*, "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories", Proc. Design Automation Conference, June 2008.

[11] Microsoft, SXM: C# Software Transactional Memory, 2005.

[12] C. Garcia, *et al.*, "Mitosis Compiler: An Infrastructure for Speculative Threading Based on Pre-Computation Slices", Proc. Conference on Programming Language Design and Implementation, 2005.

[13] Microsoft Co., "The Manycore Shift: Microsoft Parallel Computing Initiative Ushers Computing into Next Era", Microsoft white paper available at www.microsoft.com/presspass/events/supercomputing/docs/ManycoreWP.doc, Nov. 2007.

[14] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many", Intel white paper available at ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf

[15] P. Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," Intel Technology Journal, Feb. 2005.

[16] Weirong Zhu, *et al.* "Exploring Financial Applications on Many-Core-on-a-Chip Architecture: A First Experiment", Proc. ISPA Workshops 2006.

약 력

정 의 영

1988년 고려대학교 전자전산공학과 학사
1990년 고려대학교 전자공학과 석사
2002년 스탠포드 대학교 electrical engineering 박사
1990년 ~ 2005년 삼성전자 SoC 연구소 수석 연구원
2005년 ~ 현재 연세대학교 전기전자 공학부 부교수
관심분야 : 시스템 구조, 저전력 설계 기술, 플래시 메모리 응용, 바이오 응용

유 승 주

1992년 서울대학교 전자공학과 학사
1995년 서울대학교 전자공학과 석사
2000년 서울대학교 전기공학부 박사
2000년, 2001년, 2002년, 2004년 프랑스 TIMA 연구소 연구원
2001년 ~ 2002년 서울대 반도체공동연구소 연구원
2004년 ~ 2008년 삼성전자 S.LSI사업부 책임/수석연구원
2008년 ~ 현재 포항공과대학교 조교수
관심분야 : Many-core SoC architecture, low power SoC design, embedded system memory