미래의 전자 제품을 위한 최첨단 멀티코어 개발 기술

멀티코어 프로세서가 PC, 이동 전화, 게임기, 네트워크 장비, 산업용 제어 시스템과 의료기기 등의 전자 제품에 더욱 만연하게 되면서 이러한 시스템을 위한 소프트웨어의 개발 필요성이 전자 업계의 주된 관심사가 되었습니다. 멀티코어 아키텍처의 장점은 고성능, 저전력 소비, 저비용 및 뛰어난 유연성 등 매우 다양합니다. 하지만 그에 상응하는 소프트웨어가 개발되어야만 이러한 이익을 누릴 수 있습니다. 현재 전자 업계의 다수 소프트웨어 개발자들은 멀티코어에 최적화된 소프트웨어를 작성할 수 있는 능력이 부족한 실정입니다. 더욱이 멀티코어 아키텍처는 코어 수가 2개에서 4개로, 다시 32개로 늘어남에 따라 그 복잡도 역시 기하급수적으로 증가할 수 있습니다. 다시 말해 종래의 개발 방법으로는 더 이상 효과를 볼 수 없다는 것입니다. 이 복잡도를 처리할 수 있는 유일한 방법은 소프트웨어가 다양한 코어를 자동으로 배치할 수 있는 고급 언어를 사용하는 등 자동화를 통한 방법입니다. 출시 기간이 결정적 차별화 요인인 전자 산업에서, 출시 기한에 맞추기 위해서는 회사의 기존 소프트웨어 코드를 사용해야 합니다. 하지만 기존 코드가 대부분 싱글 코어 소스 코드이며 멀티코어 기술에 최적화되어 있지 않습니다.

본 IBM® Rational® 백서에서는 새로운 멀티코어 기술을 활용할 수 있는 최첨단 소프트웨어 개발 방법과 멀티코어 시스템에 싱글 코어 소프트웨어를 재사용하는 자동화 방법을 알아봅니다. 또한 기존 시스템을 재사용 함으로써 시간 및 비용을 현저히 절감하고 업계에서 선도적 위치를 다질 수 있는 방법을 살펴 보기로 합니다.

서론

소형화의 추세가 끊임없이 전자 산업에 이어지고 있습니다. 디바이스 폼팩터와 그 기본 컴포넌트들이 작아지고 있는 반면 기능과 성능은 향상되고 있습니다.

또한 디바이스는 통신 링크가 추가되면서 고도로 상호 연결된 에코시스템의 구성요소가 되고 있습니다. 전자 업체들은 자사 제품의 발전을 위해 두 개 이상의 독립 코어로 이루어진 멀티코어 프로세서[1]를 채택하고 있습니다.

멀티코어 프로세서가 PC, 이동 전화, 게임기, 네트워크 장비, 산업용 제어 시스템과 의료기기 등의 전자 제품에 더욱 만연하게 되면서 이러한 시스템을 위한 소프트웨어의 개발 필요성이 전자 업계의 주된 관심사가 되었습니다. 멀티코어 아키텍처의 장점은 고성능, 저전력 소비, 저비용 및 뛰어난 유연성 등 매우 다양합니다. 하지만 그에 상응하는 소프트웨어가 개발되어야만 이러한 이익을 누릴 수 있습니다. 현재 전자 업계의 다수 소프트웨어 개발자들은 멀티코어에 최적화된 소프트웨어를 작성할 수 있는 능력이 부족한 실정입니다. 더욱이 멀티코어 아키텍처는 코어 수가 2개에서 4개로, 다시 32개로 늘어남에 따라 그 복잡도 역시 기하급수적으로 증가할 수 있습니다. 다시 말해 종래의 개발 방법으로는 더 이상 효과를 볼 수 없다는 것입니다. 이 복잡도를 처리할 수 있는 유일한 방법은 소프트웨어가 다양한 코어를 자동으로 배치할 수 있는 고급 언어를 사용하는 등 자동화를 통한 방법입니다. 출시 기간이 결정적 차별화 요인인 전자 산업에서, 출시 기한에 맞추기 위해서는 회사의 기존 소프트웨어 코드를 사용해야 합니다. 하지만 기존 코드가 대부분 싱글 코어 소스 코드이며 멀티코어 기술에 최적화되어 있지 않습니다.

본 IBM® Rational® 백서에서는 새로운 멀티코어 기술을 활용할 수 있는 최첨단 소프트웨어 개발 방법과 멀티코어 시스템에 싱글 코어 소프트웨어를 재사용하는 자동화 방법을 알아봅니다. 또한 기존 시스템을 재사용 함으로써 시간 및 비용을 현저히 절감하고 업계에서 선도적 위치를 다질 수 있는 방법을 살펴 보기로 합니다.

멀티코어의 전망

오늘날 첨단 전자 제품의 구축에 쓰이는 컴퓨팅 플랫폼의 성능 증대와 전력 소비 및 비용 절감은 거부할 수 없는 흐름이 되었습니다. 종래에는 다이(die) 상의 트랜지스터 수를 늘리고 클록 속도를 높이는 방식으로 무어의 법칙을 초월하는 프로세서의 성능을 개선하여 왔습니다.[2] 이 추세는 계속될 것으로 보이지만 진정한 성능과 전력관리의 비약적인 발전을 가져올 방법은 멀티코어 기술에 입각한 혁신에 있습니다.

멀티코어 기반의 시스템은 전력 소비가 적으며 일반적으로 느린 클록 속도로 작동하여, 배터리 수명을 연장하고 발열을 줄일 수 있습니다. 작동 온도가 낮으면 한층 조용하고 전력 효율적인 팬리스(fanless) 방식의 냉각 시스템을 사용할 수 있습니다. 더 적은 수의 멀티코어 프로세서로 별개 프로세서의 수를 줄여, 더 작은 폼팩터의 제품을 개발할 수 있습니다. 멀티코어 운영 환경은 진정한 멀티태스킹을 보장하며 애플리케이션 성능이 싱글 코어 기반 환경에서의 것보다 우수합니다. 따라서 쉽게 CPU 이용률 임계치에 도달할 수 있습니다. 이제는 점증하는 컴퓨팅 집약적인 애플리케이션의 부하를 처리할 때 진정한 병렬 처리가 가능합니다. 멀티 코어를 이용한 부하 균형 및 시스템 기능의 분리는 시스템의 강건성과 보안의 향상을 가져올 수 있습니다.

개인용 컴퓨터 및 워크스테이션에서 하이엔드 스마트폰과 휴대형 디바이스에 이르기까지, 보다 향상된 성능, 배터리 수명 및 첨단 기능에 대한 소비자들의 요구가 높아지고 있습니다. 이에 따라 디바이스 제조업체들이 멀티코어 기술을 사용하는 제품을 개발하여 공급하기에 이르렀습니다. 차세대 개인용 전자 제품 및 스마트 제품들은 멀티코어 환경에 의존하여 보다 우수한 연결성, 응답성, 사용성 및 생산성을 갖춘 애플리케이션을 기술에 익숙한 고객들에게 제공할 것입니다. 소프트웨어는 멀티코어의 가능성을 실현하는 결정적 요인이 될 것이며 그 성공여부는 임베디드 소프트웨어 팀에 달려 있습니다.

멀티코어의 과제

멀티코어 기술의 가능성은 오늘날 제품 개발 커뮤니티에 심각한 위험과 비용을 야기합니다. 멀티코어 환경은 제품과 그 제품을 구동하는 소프트웨어 집약적인 애플리케이션의 설계 및 공급을 한층 복잡하게 만듭니다. 제품 아키텍처에 영향을 미치는, 자원과 기능의 분배에 대한 선택이 이루어져야 합니다. 시스템 설계자의 경우 실제 프로세서의 수, 필요한 코어의 수, 운영체계 (대칭 또는 비대칭) 특성 및 필요한 미들웨어에 대한 결정이 더욱 까다롭습니다. 소프트웨어 측면에서, 설계자들은 애플리케이션 파티셔닝, 인터코어(intercore) 및 인터태스크(intertask) 통신, 설계 확장성 등과 같은 멀티코어 기반 시스템에서의 새로운 문제를 고려해야 합니다. 운영체계(들) 및 잠재적 사용성을 지닌 미들웨어의 기능은 전반적으로 소프트웨어 설계의 복잡도를 심화시킵니다.

멀티코어로 이전하는 기업들은 소프트웨어 설계 팀의 스킬셋(skill set)에 변화를 기해야 할 것입니다. 병렬 처리가 표준이 되어야 하며 애플리케이션 구축 방식이 멀티코어 운영 환경에서 제공되는 병렬 구조를 충분히 이용할 수 있도록 변경되어야 합니다. 멀티 태스크의 사용 증가, 인터태스크 통신 메커니즘 및 태스크-코어(task-to-core) 할당에 관한 고려가 모두 성공적인 애플리케이션 개발에 있어 결정적인 요인이 됩니다. 재사용은 소프트웨어 집약적인 시스템의 컴포넌트와 태스크에 대한 중요성을 높입니다.

개선된 병렬 아키텍처의 성능을 근본적으로 활용하여 제 기능을 발휘하고, 새로운 멀티코어 아키텍처 상에서 실행되도록 기존 애플리케이션을 리팩토링(refactor) 해야 합니다. 싱글 코어 시스템 상에서, 나아가 멀티태스킹 운영체계에서 정확하고 안전하게 실행되었던 것도 멀티코어 시스템이 제공하는 병렬 환경에서는 오류를 보이거나 안전하지 않을 수 있습니다.

멀티코어 시스템에 소프트웨어를 배치할 때에는, 디버깅과 테스트가 더욱 더 중요해집니다. 종래의 코드 레벨 디버깅과 후반부(late-cycle) 테스트 방식은 흔히 멀티코어 환경에 적절치 않습니다. 소프트웨어가 제대로 기능하고 요구되는 수준의 기능성을 제공하는지를 확인하려면 새로운 기술 및 테스트 환경이 필요합니다.

멀티코어 개발을 위한 IBM Rational 방법

효과적인 멀티코어 개발에는 다음 세 가지 방법이 있습니다:

- 트레이드오프 조사를 실시하여 대안을 평가
- 기존 소프트웨어를 이용하여 개발을 단축
- 소프트웨어 생성을 자동화하여 품질을 개선

트레이드오프 조사를 실시하여 대안을 평가

단순히 기존 시스템을 멀티코어 프로세서로 이전하면 동일 속도 또는 느린 속도로 실행될 수 있습니다.[3] 이는 실제 코어가 개별적으로는 원래의 프로세서보다 느리고 공유 데이터가 통신을 저하시킬 수 있기 때문입니다. 심지어 모든 애플리케이션이 싱글 코어의 속도로 실행될 수도 있습니다. 대안을 평가하여 원하는 성능, 속도 및 자원 이용률 개선의 실현 여부를

확인하려면 트레이드오프 조사를 실시하는 것이 극히 중요합니다.

성공적인 구현을 위해서는 제품의 소프트웨어 아키텍처를 모델링하고 멀티 코어 상에서 실행하기 위한 다양한 옵션을 개발해야 합니다. 기존 태스크는 코어에 맵핑되며 필요 시 부가적인 태스크가 추가됩니다. 게다가 모델링을 통해 애플리케이션이 관여하는 태스크와 프로세스 간의 통신을 최적화하는 방법에 대한 이해를 제고할 수 있습니다.

또 다른 중요한 방법은 구축한 모델을 시뮬레이션 하여 모든 기능이 사전 예상과 일치하는지 확인하는 것입니다. 그러한 시뮬레이션 없이는 단지 시스템의 기능에 대한 기대로 만족하게 될 뿐입니다.

트레이드오프를 실시하면 모델을 통해 특유의 결정을 내린 이유를 문서화 할 수 있습니다. 또 다른 중요한 측면은 합의된 기존 요구사항에 설계를 맵핑하는 것입니다. 요구사항의 실시간 측면을 이해하면 각 코어에 어떤 태스크를 할당할지 수월하게 결정할 수 있습니다. 일례로, 제품 소프트웨어의 사용자 인터페이스를 실시간 또는 크리티컬한 제어 컴포넌트와 동일한 코어 상에 두지 않을 것입니다.

트레이드오프 조사는 소프트웨어의 어떤 부분을 어느 코어로 보내야 할 지에 대한 옵션을 이해하는데 매우 중요합니다. 이러한 조사 없이는 최적의 대안을 결정할 수 없으며 애플리케이션이 코어 상에서 최적으로 실행되는 지 확인할 수 없습니다.

기존 소프트웨어를 이용하여 개발을 단축

기존 제품이 이미 주요 애플리케이션 소프트웨어로 성공적으로 실행될 수 있습니다. 기존의 싱글 코어 환경에서 잘 실행되는 제품이 새로운 멀티코어 환경에서도 제 기능을 발휘할 수 있습니다. 성능을 개선하여 새로운 환경에서의 요구에 부응하고자 할 경우, 기존 소프트웨어를 최대한 재사용함으로써 그 가치를 유지해야 합니다.

모델 중심의 접근방식에 기반하여 개발된 멀티코어의 경우 기존 소프트웨어의 재사용을 충분히 가능하게 합니다. 소프트웨어 및 그 컴포넌트의 관계와 구조에 대한 간단한 시각화에서 멀티코어 환경에서의 배치 최적화를 위한 코드 리팩토링, 또는 전혀 새로운 설계와 애플리케이션 아키텍처로의 통합 및 확장에 이르기까지, 적절한 모델 중심의 도구를 갖추면 기존의 소프트웨어를 재사용할 수 있는 능력을 크게 촉진할 수 있습니다.

기존 애플리케이션 소프트웨어를 시각화하면 소프트웨어의 설계 방법과 개별 컴포넌트의 연계 방법에 대한 이해를 제고할 수 있다는 장점이 있습니다. 이로써 설계의 최선책과 차선책을 파악하고 현용 설계를 문서화하여 코드를 변경하지 않고도 새로운 설계에 사용할 수 있는 최적의 방법을 분석할 수 있습니다. 이 시각화된 코드는 새로운 멀티코어 환경에 맞출 수 있으며 트레이드오프 조사를 실시하여 코드를 재사용하거나 리팩토링하는 등, 애플리케이션을 진행시킬 최적의 방법을 판단할 수 있습니다.

기존 소프트웨어가 새로운 환경에서 제 기능을 발휘하지 못하거나 멀티코어 환경에 부적합하게 설계되었다고 판단되면 리팩토링이 유용한 옵션입니다. 코드를 모델 구성요소로 임포트하면 멀티코어에 적합한 더 나은 애플리케이션 아키텍처를 지원하도록 리팩토링하고 재생성할 수 있습니다. 모델링과 코드의 자동 생성은 기존 코드의 리팩토링을 애플리케이션 재작성에 대한 비용 효과적인 대안과도 같습니다.

마지막으로, 기존 소프트웨어 컴포넌트를 멀티코어 환경의 제품을 위하여 구축된 새로운 설계 및 애플리케이션에 사용할 수 있습니다. 다시금, 강력한 모델 기반 개발 도구의 시각화, 리팩토링 및 자동 생성 기능이 이러한 재사용 결정을 내리는 능력을 크게 강화합니다. 제품을 위한 새로운 애플리케이션을 작성하여 전달하고, 처음부터 다시 설계해야 할 필요를 없앱니다. 또한 재사용할 컴포넌트는 이전의 제품에서 성공적으로 사용된 이력이 있으므로 어느 정도는 이미 파악되고 테스트된 것입니다. 따라서 새로운 애플리케이션의 일부처럼 성능을 새로이 테스트하여 디버깅할 필요가 없습니다.

새로운 하드웨어로 이전함에 있어, 모든 것을 처음부터 작성하기에는 너무 오랜 시간이 소요되므로 레거시 코드의 재사용은 항상 중요합니다. 또한 이미 테스트된 코드를 확보하는 것은 제 때에 프로젝트를 완수하는데 있어 극히 중요합니다. 재사용할 수 있는 코드를 그래픽으로 보게 되면 새로운 멀티코어 하드웨어 플랫폼으로 전환함에 따른 추가 기능에 대한 적합성을 효과적으로 파악할 수 있습니다.

소프트웨어 생성을 자동화하여 품질을 개선

멀티코어는 본질적으로 아주 우수한 병렬 처리 환경을 제공합니다. 따라서 애플리케이션 내 태스크 간의 통신과 애플리케이션 간의 통신이 이러한 애플리케이션의 성공적인 배치에 결정적인 요인이 됩니다. 태스크는 동일한 코어 상에서 다른 코어로 전환됨에 따라, 대부분의 상황에서 다양한 메커니즘으로 통신이 이루어져야 합니다. 효율적인 통신에 대한 의존도가 아주 크기 때문에, 태스크 간의 우수한 인터페이스를 갖추고 모델링 환경을 사용하여 지원 메커니즘을 자동 생성하는 중요성이 더욱 커지고 있습니다.

다양한 하드웨어 및 통신 프로토콜로의 전환에 관한 대안을 충분히 파악하고, 명시하여 조사하려면 모델의 사용 및 구체적인 자동 코드 생성이 필요합니다. 이로써 생성된 코드 내에서 설계의 체계적인 복사 및 시각화가 가능해집니다. 태스크를 한 코어에서 다른 코어로 이전할 경우, 다른 운영체계를 사용하거나 다른 프로토콜을 사용하여 기존 코어 상의 태스크와 통신이 이루어져야 할지라도, 모델링된 변경을 재생성하여 필요 시 새로운 코어 특유의 코드에 반영할 수 있습니다. 일반적으로 이는 변경을 수작업으로 코딩해야 할 필요가 있지만 모델링된 애플리케이션의 경우, 변환의 나머지를 처리하는 모델 기반 개발 도구의 자동 생성 기능으로 간단하게 옵션을 할당하면 이 변경이 이루어질 수 있습니다. 개발의 모든 단계에서, 모델링과 코드 생성은 개발자들이 자신의 멀티코어 애플리케이션을 테스트하고 배치하는데 결정적인 역할을 합니다. 이는 전자 업계와 같이 시간에 민감한 시장에서 더욱 그러합니다.

결론

혁신적이고 차별화된 제품을 단기간 내에 출시하기 위해, 전자산업은 전반적으로 멀티코어 프로세싱을 점점 더 채택하게 될 것입니다. 이는 결과적으로 고성능, 전력 소비 저감, 긴 배터리 수명, 비용 절감 및 유연성 제고의 이익을 얻을 수 있기 때문입니다. 하지만 이러한 이익은 그에 상응하는 소프트웨어가 멀티코어에 적합하게 설계되어 있지 않다면 실현될 수 없습니다. 몇 가지 첨단 방법을 사용하여 멀티코어를 위한 소프트웨어 개발을 간소화하고 단축할 수 있습니다. 모델 기반의 개발을 통해 소프트웨어 설계 팀은 멀티코어 설계 트레이드오프 조사를 실시하고, 코드를 시각화 및 리팩토링하여 재사용률을 높입니다. 결과적으로, 사용하는 하드웨어 구성 및 통신 프로토콜에 따라 코드를 자동으로 생성할 수 있습니다.

추가 정보

멀티코어 개발 방식에 대한 자세한 정보를 알고 싶으시면 해당 IBM 영업대표 또는 Business Partner에게 문의하시거나 다음 사이트를 방문하시기 바랍니다: **ibm.com**/software/rational/info/multicore/

IBM Global Financing의 파이낸싱 솔루션은 효과적인 현금 관리, 기술적 퇴화 방지, 총소유비용 및 투자 수익률 개선 등을 가능하게 합니다. 또한 Global Asset Recovery Service는 새롭고 보다 에너지 효율적인 솔루션으로 환경 문제 해결에 기여합니다. IBM Global Financing에 관한 추가 정보가 필요하시면 다음 사이트를 방문하시기 바랍니다: **ibm.com**/financing

IBM Corporation
Software Group
Route 100
Somers, NY 10589 U.S.A.

Produced in the United States of America
March 2010

http://en.wikipedia.org/wiki/Multicore_processor

http://en.wikipedia.org/wiki/Moore's_law

http://www.forbes.com/2009/11/23/google-microsoft-programming-technology-cio-network-multicore-hardware.html

RAW14209-USEN-00