인텔® 패러렐 컴포저

제품 정보

인텔® 패러렐 컴포저

멀티 코어 시스템용 직렬 및 병렬 C 및 C++ 응용 프로그램 구축

인텔® 패러렐 컴포저는 Windows* 기반 클라이언트 응용 프로그램을 병렬화하는 개발자를 위한 인텔® C++ 컴파일러, 라이브러리 및 디버깅 기능이 포함된 종합 세트입니다. 이 제품은 Microsoft Visual Studio*와 통합되며, Visual C++과 호환되고, 개발자 작업 방식을 지원하여 IDE 투자를 보호함과 동시에 병렬 디버깅을 포함한 전례 없는 광범위한 병렬화 개발 기능을 제공합니다. 인텔 패러렐 컴포저는 독립형 제품 또는 인텔® 패러렐 스튜디오의 포함된 제품으로 구입할 수 있습니다. 인텔® 패러렐 스튜디오에는 스레딩 및 메모리 오류를 분석하기 위한 인텔® 패러렐 인스펙터와 병렬 응용 프로그램의 성능을 분석하기 위한 인텔® 패러렐 앰플리파이어가 포함됩니다.

인텔 패러렐 컴포저 컴포넌트

- 32비트 프로세서용 인텔 C++ 컴파일러, 32비트 시스템에서 64비트 응용 프로그램을 생성할 수 있는 크로스 컴파일러 및 네이티브 64비트 컴파일러
- Microsoft Visual Studio 디버거와 통합되는 인텔® 패러렐 디버거 익스텐션
- 스레드를 안정성, 이식성 및 확장성이 뛰어난 병렬 응용 프로그램을 생성하는 작업으로 추상화하는 수상 경력의 C++ 템플릿 라이브러리인 인텔® 스레딩 빌딩 블록(인텔® TBB). Visual C++과도 사용할 수 있습니다
- 인텔® 통합 성능 프리미티브(인텔® IPP)는 멀티 코어를 지원하는 광범위한 라이브러리로, 멀티미디어, 데이터 처리 및 통신 응용 프로그램에 맞게 고도로 최적화된 소프트웨어 기능을 갖추고 있습니다. 인텔 IPP에는 수작업을 통해 최적화된 기본 수준의 함수와 코덱 등 고급 스레드 솔루션이 포함됩니다. Visual C++과 .NET 개발에 모두 사용할 수 있습니다
- 개발을 신속히 시작할 수 있도록 도와줄 샘플 코드와 훌륭한 시작 안내서입니다

인텔® C++ 컴파일러

Microsoft Visual Studio 통합 및 호환성

인텔 패러렐 스튜디오의 모든 기능은 Microsoft Visual Studio 2005 및 2008에 통합됩니다. 인텔 패러렐 컴포저는 인텔 패러렐 스튜디오의 세 가지 주요 기능 그룹 중 하나입니다. 여기에는 인텔 C++ 컴파일러, 인텔 패러렐 디버거 익스텐션, 인텔 스레딩 빌딩 블록, 그리고 인텔 통합 성능 프리미티브가 포함됩니다.

이 컴파일러는 네이티브 32비트 개발, 크로스 컴파일 환경(32비트 시스템에서 64비트 응용 프로그램 개발 지원) 및 기본 64비트 개발을 제공합니다. 32비트 기능만 설치하거나, 64비트 기능만 설치하거나, 또는 두 가지 기능을 모두 설치할 수 있습니다.

인텔 C++ 컴파일러와 관련 인텔 패러렐 디버거 익스텐션은 C/C++ 개발자에게 많은 장점을 제공하지만, 인텔 패러렐 컴포저 또는 전체 인텔 패러렐 스튜디오에 포함된 다른 컴포넌트를 반드시 사용할 필요는 없습니다. 다시 말해서 인텔 스레딩 빌딩 블록과 인텔 통합 성능 프리미티브를 Visual C++ 컴포저와 함께 사용할 수 있다는 뜻입니다. 또한 Visual C++로 작성된 응용 프로그램에서 인텔 패러렐 스튜디오 메모리 누수 또는 병행성 검사 기능도 사용할 수 있습니다. 한마디로, 강력하고 사용하기 쉬운 호환 인텔 C++ 컴파일러를 비롯하여, 전체 인텔 패러렐 스튜디오의 사용과 관련하여 개발자의 흥미를 끌만한 요소가 많이 있습니다.

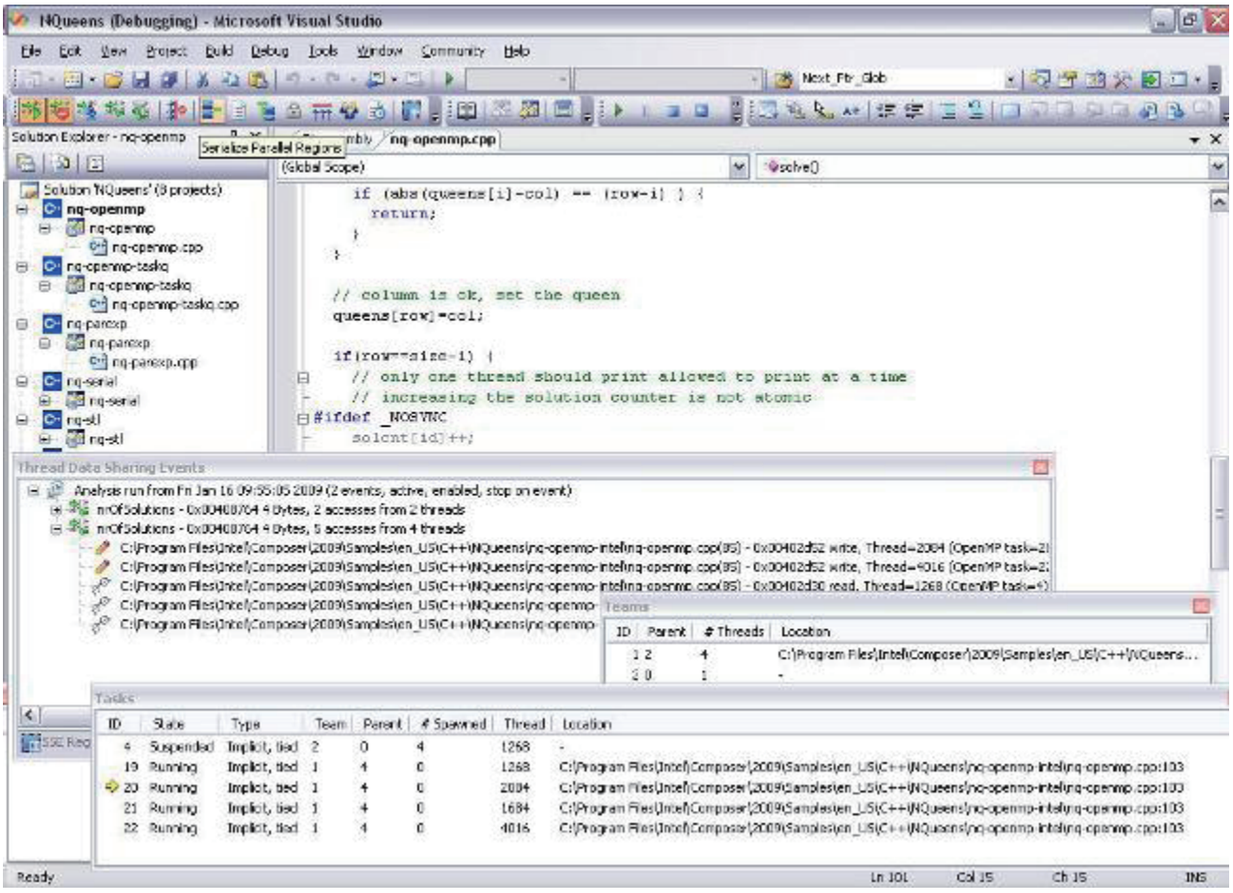

그림 1: 인텔® 패러렐 컴포저는 Visual Studio*에 통합됩니다. 화면의 솔루션은 인텔 C++을 사용합니다. 프로젝트(Project) 메뉴를 통해 또는 솔루션이나 프로젝트 이름 위에서 마우스 오른쪽 단추를 눌러 Visual C++로 쉽게 전환할 수 있습니다

쉽게 시작할 수 있으며 성장하고 있는 병렬화 커뮤니티와의 지속적인 관계 유지

인텔 패러렐 컴포저에는 방법에 대한 설명을 전달하는 데 사용되는 기능 둘러보기와 코드 샘플을 제공할 사용하기 쉬운 시작 안내서가 포함되어 있습니다. 또한 간단한 비디오를 볼 수 있는 링크도 포함되어 있어 인텔 패러렐 컴포저의 병렬화 기능을 사용하는 방법에 대해 걱정할 필요가 없습니다. 인텔 패러렐 컴포저의 사용자는 이 안내서가 잠시 시간을 내어 읽어볼 만한 충분한 가치를 가지고 있다고 말합니다. 또한 도구를 생산적으로 사용할 수 있도록 해주는 병렬화 개념 및 기법을 소개하는 부분에서 샘플 코드가 매우 유용했다고 말합니다.

시작 안내서는 몇몇 위치에서 제공됩니다. 예를 들면, 인텔 웹 페이지, 설치 시 몇몇 지점, Visual Studio 도움말 메뉴(상세 설명서와 함께) 또는 Windows "시작" 단추에서 사용할 수 있는 인텔 패러렐 스튜디오 또는 인텔 패러렐 컴포저 트리 구조에서 찾아볼 수 있습니다. 설치 완료 시에도 시작 안내서를 읽어볼 것인지 묻는 메시지가 표시됩니다. 시작 안내서는 초보자든 병렬화 전문가든, 누구라도 잠시 시간을 내어 읽어볼 만한 충분한 가치를 지니고 있습니다.

시작하고 나면 성장하고 있는 개발자 커뮤니티에 참여하여 인텔 멀티 코어 프로세서 기반 시스템을 활용하는 개발자들과 정보를 교환할 수 있음이 매우 유용하게 생각될 것입니다. 인텔은 개발자가 아이디어를 교환하고 의견 및 질문을 게시하며 포인트를 획득하여 인텔® 블랙 벨트 소프트웨어 개발자가 될 수 있는 동적 포럼을 제공합니다. 또한 병렬화에 관심이 있는 개발자를 위해 계속 늘어나고 있는 방대한 기술 자료도 제공합니다. 지금 바로 커뮤니티에 참여하십시오. 병렬 프로그래밍 및 멀티코어 커뮤니티 (http://software. intel.com/en-us/articles/intel-parallel-studio/)를 방문하십시오. 여기서는 블로그, 기술 자료, 다운로드 등 모든 리소스를 찾아볼 수 있습니다. 자유롭게 살펴본 다음, 즐겨찾기 목록에 저장하는 것도 잊지 마십시오.

람다 함수 지원

인텔 컴파일러는 다음 C++ 표준인 C++0x의 초안을 지원하여 람다 함수를 구현하는 최초의 C++ 컴파일러입니다. 람다 구조는 C++의 함수 개체 또는 C의 함수 포인터와 거의 동일합니다. 코드와 스코프를 결합하기 때문에 클로저와 함께 강력한 개념을 나타냅니다. 클로저는 텍스트 형태로 함수 정의를 포함하는 양식을 바인딩함으로써 설정되는 바인딩 값을 참조하고 변경할 수 있는 함수입니다. 간단히 말해 람다 함수와 클로저는 함수 개체 또는 람다를 작성할 수 있는 편리한 방법을 제공하는 함수 개체 및 함수 포인터와 관련된 문법적 편의로 볼 수 있습니다.

그림 2의 소스 코드는 람다 식으로 만든 함수 개체의 예입니다. C++와 인텔 TBB를 더욱 밀접하게 통합하면 람다 함수와 클로저를 사용해 코드를 매개변수로 넘김으로써 함수자(functor) operator() 개념을 간소화할 수 있습니다.

```
void solve() {
  parallel_for(blocked_range<size_t>(0, size, 1),
  [](const blocked_range<int> &r){
    for (int i = r.begin(); i != r.end(); ++i)
    setQueen(new int[size], 0, (int)i);
    };
  });
}
```

그림 2: 람다 함수의 소스 코드 예

단순 병행 함수

인텔 패러렐 스튜디오에 포함된 인텔® C++ 컴파일러는 네 개의 새로운 키워드(__taskcomplete, __task, __par 및 __critical)를 제공하여 병렬 프로그래밍을 더욱 쉽게 만듭니다. 이 키워드들로 인해 가능해진 병렬성으로부터 응용 프로그램이 혜택을 얻게 하려면 /Qopenmp 컴파일러 옵션을 지정한 후 다시 컴파일해 병렬성의 실제 정도를 관리하는 적절한 런타임 지원 라이브러리에 연결합니다. 이 새로운 키워드들은 OpenMP 3.0* 런타임 라이브러리를 사용해 병렬성을 제공하지만, OpenMP* 프라그마와 지시자 문법을 가지고 이를 실제로 표현할 필요는 없게 만듭니다. 이를 통해 C 또는 C++에서 작성된 코드를 더욱 자연스럽게 유지할 수 있습니다.

앞에서 언급한 키워드는 구문 접두사로 사용됩니다. 예를 들어, __par를 사용해 solve() 함수를 병렬화할 수 있습니다.

인수 사이에 겹치는 부분이 없다고 가정하면 solve() 함수는 __par 키워드를 추가해 변경됩니다. 함수가 호출되는 방식에 대한 변경 없이 계산이 병렬화됩니다. 그림 3은 이를 보여주는 예입니다.

```
void solve() {
  __par for(int i=0; i<size; i++) {
  // try all positions in first row
  // create separate array for each recursion
  // started here
  setQueen(new int[size], 0, i);
  }
}
```

그림 3: 인텔® 패러렐 스튜디오의 인텔 C++ 컴파일러에 새롭게 추가된 4개의 단순 병행 함수 중 하나인 __par의 예

OpenMP 3.0

OpenMP는 이식 가능한 멀티 스레드 응용 프로그램 개발의 업계 표준입니다. OpenMP는 파인-그레인(fine-grain: 루프 레벨) 및 대형-그레인(large-grain: 함수 레벨) 스레딩에 효과적입니다. OpenMP 3.0은 지시자 접근 방식을 사용해 데이터 및 태스크 병렬성을 모두 지원함으로써 직렬 응용 프로그램을 병렬 응용 프로그램으로 변환하는 쉽고 강력한 방법을 제시하며, 잠재적으로 멀티 코어 및 대칭 멀티 프로세서 시스템에서 병렬 실행을 통해 성능을 크게 향상시킬 수 있습니다.

OpenMP를 사용해 작성하고 구축한 응용 프로그램이 단일 프로세서 시스템에서 실행되면 결과는 변경되지 않은 소스 코드와 동일합니다. 다시 말하면, 결과는 변경되지 않는 직렬 실행 코드와 동일합니다. 따라서 직렬 일관성을 유지하는 동시에 더 쉽게 점진적 코드 변경을 수행할 수 있습니다. 지시자만 코드에 삽입될 수 있기 때문에 단일 프로세서 시스템에서 실행할 때 점진적 코드 변경을 수행하고 소프트웨어에 대한 공통 코드 기반을 유지하는 것이 가능합니다.

OpenMP는 멀티 플랫폼 및 운영 체제를 지원하는 단일 소스 코드 솔루션입니다. 또한 OpenMP 런타임이 올바른 숫자를 선택하기 때문에 코어의 수를 응용 프로그램에 "하드 코딩" 할 필요가 없습니다.

OpenMP 3.0태스크 큐잉

불규칙한 패턴의 동적 데이터나 재귀 같은 복잡한 제어 구조를 갖는 프로그램은 효율적인 방법으로 병렬화하기가 종종 어렵습니다. OpenMP 3.0의 작업 큐잉 모델을 사용하면 OpenMP 2.0 또는 2.5에서 가능한 것 이상으로 불규칙한 병렬화를 이용할 수 있습니다.

태스크 프라그마는 작업 단위(태스크)가 실행되는 환경을 지정합니다. 태스크 프라그마를 만나면 태스크 블록 내 코드는 태스크와 관련된 큐 속으로 큐잉됩니다. 연속되는 의미를 보존하기 위해 태스크의 완료 지점에 암시적 장벽이 있습니다. 개발자는 태스크 블록 간에 또는 태스크 블록 내 코드와 해당 태스크 블록 밖에 있는 태스크 블록의 코드 간에 디펜던시가 존재하지 않게 하거나 디펜던시가 적절하게 동기화되게 할 책임이 있습니다. 그림 4는 이를 보여주는 예입니다.

```
#pragma omp parallel
#pragma omp single
{
  for(int i=0; i<size; i++) {
    // try all positions in first row
    // create separate array for each recursion
    // started here
#pragma omp task
    setQueen(new int[size], 0, i);
  }
}
```

그림 4: OpenMP3 3.0 태스크 큐잉의 예

그림 4의 예에서는 단 하나의 태스크 큐가 필요합니다. 따라서, 단 하나의 스레드(omp single)를 사용해 큐를 설정해야 합니다. setQueens 호출은 서로 독립적이기 때문에 태스크 개념에 잘 들어맞습니다. 또한 전용 창의 OpenMP 프로그램에서 tasks, teams, locks, barriers, taskwaits의 상태 검사를 쉽게 만드는, 바로 아래에 소개된 인텔 패러렐 디버거 익스텐션에 대한 설명을 읽어보는 것도 좋습니다.

인텔 패러렐 디버거 익스텐션

인텔 패러렐 컴포저는 설치 후 Visual Studio의 디버그(Debug) 풀 다운 메뉴를 통해 액세스할 수 있는 인텔® 패러렐 디버거 익스텐션을 포함합니다(그림 5 참조).

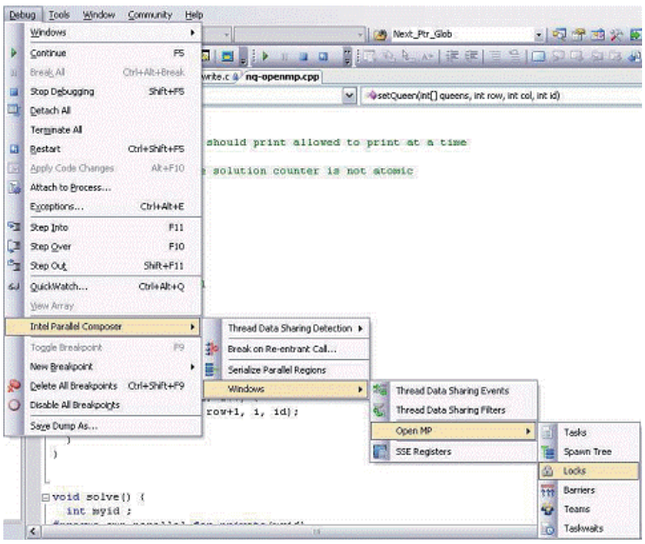

그림 5: 인텔 패러렐 디버거 익스텐션은 Microsoft Visual Studio의 디버그(Debug) 풀 다운 메뉴를 통해 액세스가 가능합니다

인텔 패러렐 디버거 익스텐션은 병렬 응용 프로그램에서 공유 데이터와 데이터 디펜던시에 대한 추가 정보와 액세스를 제공합니다. 이를 통해 개발 주기가 더욱 빨라지고 심각한 런타임 문제를 일으킬 수 있는 잠재적 데이터 액세스 충돌을 조기에 발견할 수 있습니다. 인텔 패러렐 컴포저를 설치하고 Visual Studio를 시작하고 나면 응용 프로그램이 SIMD(Single Instruction Multiple Data) 실행을 활용할 때마다 인텔 패러렐 디버거 익스텐션을 사용할 수 있고, 병렬화된 응용 프로그램이 OpenMP 스레딩을 사용할 경우 실행 흐름과 잠재적 런타임 충돌을 파악할 수 있습니다.

공유 데이터 이벤트 발견, 함수 재진입 발견, 병렬화된 코드의 직렬화된 실행을 포함한 OpenMP 인식 같은 인텔 패러렐 디버거 익스텐션의 고급 기능을 활용하려면 /debug:parallel 옵션을 사용해 인텔 컴파일러로 코드를 컴파일합니다.

자세한 내용은 인텔 패러렐 디버거 익스텐션 백서 (http://software.intel.com/en-us/articles/parallel-debugger¬extension/) 를 참조하십시오. 이 백서는 인텔 디버거 익스텐션에 대한 더욱 상세한 정보와 디버거 익스텐션이 제공하는 혜택, 디버거 익스텐션의 활용 방법 등을 제공합니다.

병렬 응용 프로그램 관련 디버깅 제품을 평가하려면 규모가 큰 인텔 패러렐 스튜디오 제품을 고려해 보십시오. 여기에는 메모리 누수 분석 및 스레드 검사 도구를 제공하는 인텔 패러렐 인스펙터가 포함되어 있습니다. 또한 핫스팟(성능 병목 지점) 분석과 병행성 검사 도구를 제공하는 인텔 패러렐 앰플리파이어도 포함되어 있어 병렬화된 코드 및 데이터를 인식하고 코드를 디버깅하여 정확성을 유지할 수 있습니다. 인텔 패러렐 스튜디오는 인텔 패러렐 디버거 익스텐션을 비롯하여 이러한 모든 기능을 제공합니다.

난해한 병렬 루프 최적화

반복 독립성을 갖는 데이터 병렬성을 표시하는 알고리즘은 "난해한 병렬" 코드를 보여주는 루프에 적합합니다. 인텔 패러렐 컴포저는 최소한의 노력으로 이러한 루프의 성능을 극대화하는 세 가지 방법을 제시합니다. 그 세 가지 방법은 자동 벡터화, 인텔 최적화 valarray 컨테이너의 사용 그리고 자동 병렬화입니다. 인텔 패러렐 컴포저는 자동 벡터화에 적합한 루프를 자동으로 찾아냅니다. 여기에는 정적 또는 동적 배열, 백터 및 valarray 컨테이너가 있는 명시적 루프 또는 명시적 루프가 있는 사용자 정의 C++ 클래스가 포함됩니다. 특별한 경우로서 암시적 valarray 루프가 자동 백터화되거나 인텔 통합 성능 프리미티브(IPP)의 라이브러리 프리미티브를 실행하도록 지시될 수 있습니다. 자동 백터화 및 최적화된 valarray 헤더의 사용은 응용 프로그램의 성능을 최적화해 스트리밍 SIMD 익스텐션을 지원하는 프로세서를 완벽하게 활용합니다.

잠시 후에 인텔 최적화 valarray 헤더의 활성화 방법을 살펴보겠습니다. 그러나 먼저 명시적 valarray, 백터 루프 및 암시적 valarray 루프의 예를 보여주는 그림 6을 살펴보겠습니다.

```
valarray<float> vf(size), vfr(size);
vector<float> vecf(size), vecfr(size);


//log function, vector, explicit loop
for (int j = 0; j < size-1; j++) {
  vecfr[j]=log(vecf[j]);

}

//log function, valarray, explicit loop
for (int j = 0; j < size-1; j++) {
  vfr[j]=log(vf[j]);
}

//log function, valarray, implicit loop
vfr=log(vf);
```

그림 6: 위의 소스 코드는 명시적 valarray, 백터 루프 및 암시적 valarray 루프의 예를 보여줍니다

최적화된 valarray 헤더를 사용하려면 인텔 통합 성능 프리미티브의 사용을 빌드 컴포넌트 선택(Build Component Selection)으로 지정하고 명령줄 옵션을 설정해야 합니다. 이를 위해 먼저 프로젝트를 Visual Studio로 불러온 후 프로젝트 속성 팝업 창을 엽니다.
"추가 옵션(Additional Options)" 상자에서 "/Quse-intel-optimized-headers"를 추가하고 "확인(OK)"을 클릭합니다. 그림 7은 이 방법을 보여주는 예입니다.

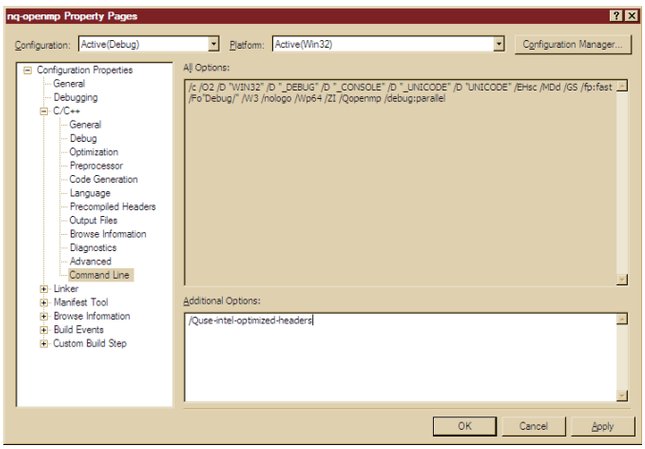

그림 7: 최적화된 헤더 파일의 사용을 위해 명령어를 Visual C++의 명령 라인에 추가하기

다음으로 프로젝트(Project) 메뉴에서 빌드 컴포넌트 선택(Build Component Selection) 팝업을 엽니다. "인텔 통합 성능 프리미티브(Intel Integrated Performance Primitives)"의 오른쪽에 있는 상자에서 "공통(Common)"을 선택한 후 "확인(OK)"을 클릭합니다. 그림 8은 이를 보여주는 예입니다. 그러고 나면 응용 프로그램을 재구축할 수 있고, 응용 프로그램을 변경할 때와 마찬가지로 성능과 행동을 검사할 수 있습니다.

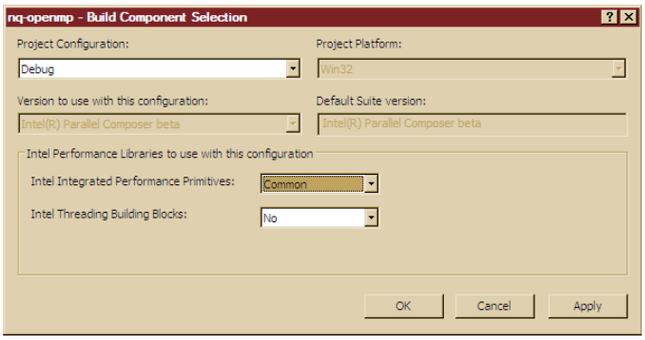

그림 8: Visual Studio에 인텔 IPP를 사용하라고 지시

자동 병렬화는 병렬로 안전하게 실행할 수 있는 병렬 루프를 찾고 멀티스레드 코드를 자동으로 생성하여 멀티 코어 프로세서를 활용함으로써 응용 프로그램 성능을 향상시킵니다. 자동 병렬화를 사용하면 사용자가 반복 파티셔닝, 데이터 공유, 스레드 스케줄링 및 동기화와 같은 하위 레벨의 세부 작업을 처리할 필요가 없습니다.

자동 병렬화는 자동 벡터화를 수행하고 최적화된 valarray 헤더를 사용하여 SSE를 지원하는 멀티 코어 시스템에서 성능을 최대화합니다. 멀티스레드 응용 프로그램 지원에 대한 자세한 내용은 사용자 안내서 (http:// software.intel.com/en-us/intel-parallel-composer/로 이동한 후 설명서(documentation) 링크 클릭)를 참조하십시오.

인텔 스레딩 빌딩 블록

스레드를 안정성, 이식성 및 확장성이 뛰어난 병렬 응용 프로그램을 생성하는 작업으로 추상화하는 수상 경력의 C++ 템플릿 라이브러리인 인텔 스레딩 빌딩 블록(인텔 TBB). 인텔 패러렐 컴포저에 포함된 인텔 TBB는 인텔 C++ 컴파일러 또는 Microsoft Visual C++와 함께 사용할 수 있는 표준 템플릿 라이브러리(STL)입니다.

인텔 TBB는 병렬 프로그래밍에 대한 세 가지 주요 문제를 해결합니다.

- **생산성:** 병렬화 구현 간소화
- **정확성:** 병렬 동기화 문제 제거
- **유지보수:** 현재뿐만 아니라 미래의 응용 프로그램 개발도 지원

인텔 TBB 사용 시 이점:

- 미래 보장형(Future-proof) 응용 프로그램: 코어 수와 스레드 수가 증가함에 따라, 인텔 TBB의 정교한 태스크 스케줄러를 사용하여 응용 프로그램 속도도 향상됩니다.
- 이식성: 병렬화를 한 번만 구현하여 여러 플랫폼에서 스레드 코드를 실행할 수 있습니다.
- 상호 운용성: 다양한 스레딩 방법, 하드웨어 및 운영 체제에서 작동합니다.
- 활성화된 오픈 소스 커뮤니티: 인텔 TBB는 오픈 소스 버전으로도 제공됩니다. opentbb.org는 포럼, 블로그, 코드 샘플 등을 제공하는 활성화된 사이트입니다.

인텔 TBB는 병렬화를 위한 포괄적이고 추상화된 템플릿, 컨테이너 및 클래스를 제공합니다. 그림 9는 인텔 TBB 내 주요 기능 그룹을 보여줍니다.

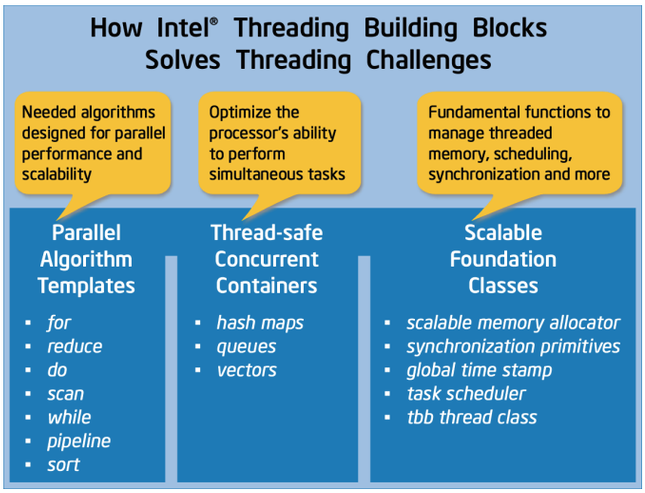

그림 9: 인텔® TBB 내 주요 기능 그룹

문제	솔루션
병렬화를 쉽게 추가하는 방법	인텔 TBB parallel_for 명령 간단한 for/next 루프 대체를 통한 병렬화 이점 활용 루프 반복 수와 상관 없이 고정된 횟수의 로드 밸런싱이 유지된 병렬 실행
확장성을 최대화하기 위한 스레드 관리	인텔 TBB 태스크 스케줄러 스레드 풀을 관리하고 네이티브 스레드의 복잡성 제거 병렬 프로그래밍의 일반적인 성능 문제를 해결하도록 설계됨 - 초과 가입(Oversubscription): 하드웨어 스레드당 스케줄러 스레드 한 개 - 높은 오버헤드: 프로그램가 스레드가 아닌 태스크 지정 - 로드 불균형: 작업 가로채기(Work-stealing)를 통한 로드 밸런싱 유지
메모리 할당은 병행 환경에서 병목 지점	인텔 TBB는 스레딩 메모리 관리 알고리즘에 따라 테스트 및 조정된 확장 가능한 메모리 할당자를 제공합니다. - STL 템플릿 클래스의 할당자 인수로 - malloc/realloc/free 호출 대용으로(C 프로그램) - 전역 새 연산자 및 삭제 연산자 대용으로(C++ 프로그램)

그림 10: 인텔® TBB는 세 가지 주요 병렬화 문제를 해결합니다

인텔 통합 성능 프리미티브(인텔 IPP)

인텔 패러렐 컴포저에는 인텔 통합 성능 프리미티브가 포함되어 있습니다. 인텔 IPP는 멀티미디어, 데이터 처리, 통신 응용 프로그램을 위해 멀티 코어에 최적화된 소프트웨어 함수로 구성된 광범위한 라이브러리입니다. 인텔 IPP는 비디오 코딩, 신호 처리, 오디오 코딩, 이미지 처리, 음성 코딩, JPEG 코딩, 음성 인식, 컴퓨터 영상, 데이터 압축, 이미지 색상 변환, 크립토그라피, 스트링 처리/규칙적 식, 벡터/매트릭스 수학 등에서 자주 사용되는 기본 알고리즘을 포함하는 최적화된 수천 개의 함수를 제공합니다.

인텔 IPP에는 수작업을 통해 최적화된 기본 수준의 함수와 코덱 등 고급 스레드 샘플이 모두 포함되어 있으며 Visual C++ 및 .NET 개발용으로 사용할 수 있습니다. 이러한 모든 함수와 샘플은 스레드에 안전하며 대다수가 내부적으로 스레딩되어 있어 현재의 멀티 코어 프로세서를 최대로 활용하고 향후 매니코어(manycore) 프로세서까지 확장할 수 있습니다.

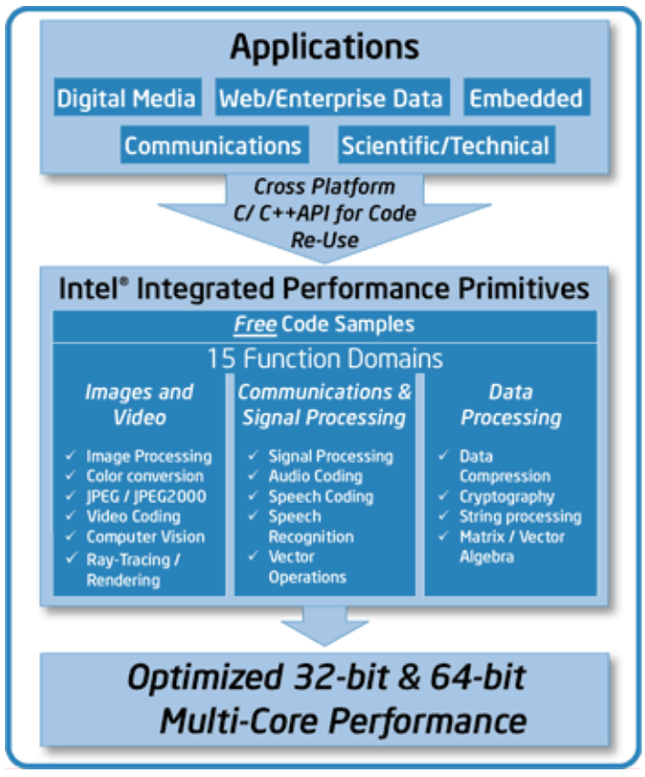

그림 11: 인텔® 통합 성능 프리미티브는 인텔® 패러렐 스튜디오의 일부인 인텔® 패러렐 컴포저에 포함되어 있으며, 위에 나열된 다양한 도메인상에서 스레딩되고 스레드에 안전한 라이브러리 함수를 제공합니다

인텔 IPP 성능

응용 프로그램 및 워크로드에 따라 인텔 IPP 함수는 컴파일된 해당 C 코드보다 더 빨리 여러 번 실행할 수 있습니다. 아래에 나오는 이미지 크기 조정 예를 보면, 컴파일된 C++ 코드에서 실행할 때는 338 ms(마이크로초)가 걸렸지만, 인텔 IPP 이미지 처리 함수를 사용했을 때는 111 ms 밖에 걸리지 않았습니다. 즉 성능이 300% 향상된 셈입니다.

그림 12: 이 이미지 조정 예에서(256x256비트에서 460x332비트로) 인텔® IPP 사용 응용 프로그램의 경우 실행 시간이 111 ms인 것에 반해, 컴파일된 C++ 코드의 경우에는 실행 시간이 338 ms이었습니다(시스템 구성: 인텔® 제온, 2.9 GHz, 2개 프로세서, 프로세서당 4개 코어, 프로세서당 2개 스레드)

Visual Studio에서 인텔 IPP 사용

Microsoft Visual Studio 프로젝트에 인텔 IPP 지원을 쉽게 추가할 수 있습니다. 인텔 패러렐 컴포저에는 인텔 IPP 라이브러리 이름 및 경로를 Visual Studio 프로젝트에 추가할 수 있는 메뉴와 대화 상자가 포함되어 있습니다. 단순히 솔루션 탐색기(Solution Explorer)에서 프로젝트 이름을 클릭하고 인텔 빌드 컴포넌트 선택(Intel Build Components Selection) 메뉴 항목을 선택한 후, 빌드 컴포넌트(Build Components) 대화 상자를 사용하여 인텔 IPP를 추가합니다. 그런 다음에는 다음 헤더와 작동 코드를 포함하여 프로젝트에 인텔 IPP 코드를 추가하기만 하면 됩니다. 빌드 선택(Build Selection) 대화 상자에서 자동으로 라이브러리 이름을 IPP용 링커에 추가하고 인텔 IPP 라이브러리에 경로를 추가합니다.

C++ 프로젝트 외에도, 포함된 랩퍼 클래스를 사용하는 C# 프로젝트에서 문자열 처리, 이미지 처리, 신호 처리, 색상 변환, 크립토그라피, 데이터 압축, JPEG, 매트릭스 및 벡터 수학과 관련하여 C#에서의 인텔 IPP 함수 호출을 지원하는 데 사용할 수 있습니다.

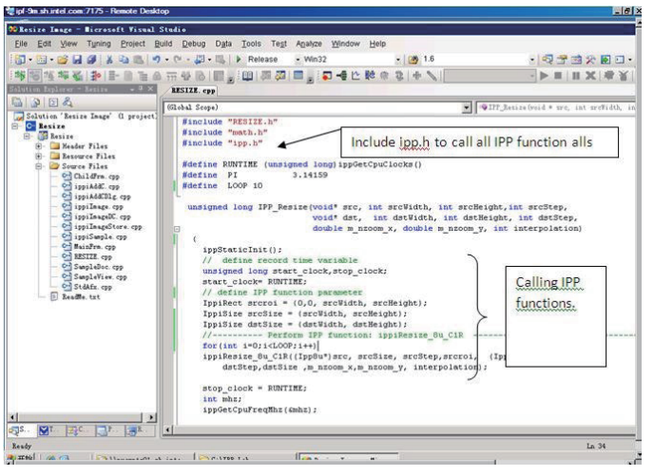

그림 13: 인텔® IPP 라이브러리 호출을 Visual Studio* 코드에 쉽게 통합할 수 있습니다

기능

- C/C++ 병렬화를 지원하도록 Microsoft* Visual Studio를 완벽하게 업그레이드합니다. Visual Studio로 통합되어 IDE 투자를 보호함과 동시에 병렬화 기능을 추가합니다.
- 인텔 패러렐 디버거 익스텐션은 Microsoft 디버거와 통합되어 병렬화 문제를 쉽게 찾아서 해결할 수 있도록 Visual Studio를 향상시킵니다. 응용 프로그램을 사용 가능한 상태로 만드는 데 드는 시간이 단축됩니다.
- 단순 병행 함수, 데이터 병렬 배열 및 수천 개의 스레드 라이브러리 함수가 포함되어 스레드 태스크를 간소화하고 응용 프로그램 개발을 가속화합니다.
- 자동 병렬화와 자동 벡터화 옵션으로 개발을 간소화하고 시간을 절약합니다.
- 통합된 배열 표기, 데이터 병렬 인텔 IPP 함수가 오디오, 비디오, 신호 분석 및 기타 응용 프로그램 클래스의 속도를 향상시킵니다.
- 병렬 응용 프로그램을 구현하고 멀티코어 플랫폼을 최대로 활용할 수 있는 가장 효율적인 방법인 인텔 TBB가 포함되어 있습니다.
- 병렬화 시작 방법을 설명하는 코드 샘플 등 자세한 설명서가 수록되어 있습니다. 또한 단 몇 분만에 시작할 수 있도록 도와줄 간단한 시작 안내서도 포함되어 있습니다.
- 커뮤니티를 지원합니다. 여러분은 더 이상 혼자가 아닙니다. 코드에 병렬화를 추가하는 개발자들로 구성된 성장하고 있는 개발자 커뮤니티에 참여할 수 있습니다. 다른 개발자의 경험을 듣고 자신의 지식과 경험을 알려주며, 적극적인 커뮤니티 활동에 대한 보상도 받을 수 있습니다.

시스템 요구사항

- Microsoft Visual Studio
- 최신 시스템 요구사항에 대한 자세한 내용은 다음 웹 사이트를 참조하십시오 : www.intel.com/software/products/systemrequirements/

지원

인텔 패러렐 스튜디오 제품은 커뮤니티 포럼에 대한 액세스와 기술 노트, 응용 프로그램 노트, 문서 및 모든 제품 업데이트 등 기술 지원을 위해 필요한 여러 가지 기술 자료를 제공합니다.

자세한 내용은 다음 사이트를 참조하십시오.
http://software.intel.com/en-us/articles/intel-parallel-studio/

베타 버전 출시

응용 프로그램에 병렬화를 추가하여 오늘날의 시장에서 증가하고 있는 멀티 코어 시스템의 설치 기반을 활용하고 추후 등장할 매니코어(manycore) 시스템에 맞춰 응용 프로그램의 미래 경쟁력을 확보할 수 있습니다.

다운로드 및 사용자 포럼 등록에 대한 자세한 내용은 다음 웹 사이트를 참조하십시오 : www.intel.com/software/ParallelStudioBeta/

인텔® 패러렐 스튜디오

현재의 직렬 응용 프로그램 및 미래의 혁신적인 소프트웨어 개발자를 위해 설계되었습니다.

인텔은 멀티 코어용 직렬 및 새로운 병렬 응용 프로그램과 매니코어(manycore)용 스케일 조정을 최적화하도록 설계된 완전한 프로덕션 솔루션을 통해 Microsoft Visual Studio* C++ 개발자에게 간소화된 병렬화를 제공합니다.

인텔® 패러렐 스튜디오:최상의 올인원 병렬화 툴킷을 사용하여 최적화된 직렬 및 병렬 응용 프로그램을 작성합니다.

인텔® 패러렐 컴포저: C/C++ 컴파일러 및 고급 스레드 라이브러리로 효과적인 응용 프로그램을 개발합니다.

인텔® 패러렐 인스펙터: 사전 메모리 및 스레딩 오류 검사를 통해 응용 프로그램의 안정성을 유지합니다.

인텔® 패러렐 앰플리파이어: 멀티 코어의 성능 개선을 위해 병목 현상을 신속하게 발견하고 병렬 응용 프로그램을 조정합니다.

0209/BLA/CMD/PDF 321554-001