인텔® 패러렐 스튜디오

제품 정보

인텔® 패러렐 스튜디오

개발 수명 주기를 위한 병렬성

인텔® 패러렐 스튜디오는 C/C++ Microsoft Visual Studio* 응용 프로그램 개발에 포괄적인 병렬성을 제공합니다. 인텔 패러렐 스튜디오는 소프트웨어 업계와 개발자들의 염려를 반영해 개발되었습니다. 제품들이 각자의 고유한 기능에 맞게 개발 수명 주기를 지원하도록 상호 연동하기 때문에 병렬성을 전보다 훨씬 쉽게 구현할 수 있게 되었습니다. 이러한 특징으로 인해 인텔 패러렐 스튜디오를 사용하면 병렬성을 처음 접하는 개발자들도 쉽게 배울 수 있고 숙련된 프로그래머들도 더욱 효율적이고 신뢰성 높게 작업할 수 있습니다. 인텔 패러렐 스튜디오는 인텔® 스레딩 빌딩 블록과 OpenMP* 같은 공통 병렬 프로그래밍 라이브러리 및 API 표준과 상호 연동하며, 멀티 코어 플랫폼의 장점을 즉각적으로 구현할 수 있게 해줍니다.

"인텔® 패러렐 스튜디오의 새로운 분석 및 프로파일링 도구는 새로운 Envivio 4Caster* 시리즈 트랜스코더의 개발을 더욱 빠르고 더욱 효율적으로 만듭니다. 특히, 인텔® 패러렐 인스펙터와 인텔® 패러렐 앰플리파이어를 사용하면 멀티 코어, 멀티 스레드 환경에서 코드의 신뢰성과 성능을 개선해 전체 소프트웨어 개발 시간을 단축시킬 수 있습니다. 검증 단계에서 기능 장애의 수는 더욱 안전한 실행 덕분에 감소하며, 버그 추적도 쉬워집니다. 인텔 패러렐 스튜디오는 우리 소프트웨어 제품의 시장 출시를 전반적으로 앞당깁니다."

에릭 로시어(Eric Rosier)
Envivio
엔지니어링 부사장

인텔® 패러렐 스튜디오 도구

인텔® 패러렐 앰플리파이어:

멀티 코어의 성능 개선을 위해 병목 현상을 신속하게 발견하고 병렬 응용 프로그램을 조정합니다

- 응용 프로그램 핫스팟을 발견하고 소스 코드를 분석합니다
- 병행성 분석을 통한 성능 개선을 위해 병렬 응용 프로그램을 조정합니다
- 잠금 및 대기 분석을 사용해 병렬 성능을 제한하는 중대 지연을 찾아냅니다
- 결과 비교를 통해 변경 사항이나 성능 저하를 빠르게 파악할 수 있습니다

인텔® 패러렐 컴포저:

C/C++ 컴파일러 및 고급 스레드 라이브러리로 효과적인 응용 프로그램을 개발합니다.

- 32비트 프로세서용 인텔® C++ 컴파일러로 구축하면 32비트 시스템에서 64비트 응용 프로그램을 생성할 수 있는 크로스 컴파일러로도 활용할 수 있고, 기본 64비트 컴파일러로도 활용할 수 있습니다.

- 엔지니어링, 금융, 디지털 미디어, 데이터 처리, 수학 등의 다양한 분야에서 사용되는 기본 레벨의 빌딩 블록인 인텔® 통합 성능 프리미티브(인텔® IPP)로 코딩할 수 있습니다. 또한 인텔® IPP는 Microsoft Visual C++ 컴파일러와 함께 사용될 수 있습니다

- Microsoft Visual Studio 디버거와 통합되는 인텔® 패러렐 디버거 익스텐션으로 디버깅할 수 있습니다.

- 믿을 수 있고 이식 가능하며 확장 가능한 병렬 응용 프로그램 생성 작업 시 스레드를 추출하는 수상 경력의 C++ 템플릿 라이브러리인 인텔® 스레딩 빌딩 블록(인텔® TBB)으로 코딩할 수 있습니다. 또한 인텔® TBB는 Microsoft Visual C++ 컴파일러와 함께 사용될 수 있습니다.

인텔® 패러렐 인스펙터:

사전 대응적인 병렬 메모리 및 스레딩 오류 검사로 응용 프로그램 신뢰성을 보장합니다.

- 데드락이나 데이터 레이스 같은 스레딩 관련 오류를 찾아냅니다
- 메모리 누수나 훼손 같은 메모리 오류를 찾아냅니다

그림 1. 인텔® 패러렐 스튜디오 작업

인텔® 멀티 코어 프로세서를 완벽하게 활용하고 멀티 코어 아키텍처에서 최대의 응용 프로그램 성능을 달성하려면 스레드를 효과적으로 사용해 소프트웨어 워크로드를 분산해야 합니다. 효과적인 병렬 응용 프로그램을 생성하기 위해 코드에 스레드를 추가할 때 일반적으로 다음과 같은 질문에 직면하게 됩니다:

a. 최대의 성능 개선을 달성하고 메모리 충돌을 피하기 위해 병렬화를 구현할 때 응용 프로그램의 어느 부분이 가장 적절한가?

b. 응용 프로그램에 가장 적절한 프로그래밍 모델 및 특정 스레딩 방법은 무엇인가?

c. 스레드 소프트웨어가 비결정적인 방법으로 실행되어 실행 시퀀스가 해당 소프트웨어의 실행에 의존하기 때문에 재현이 어려운 스레딩 및 메모리 오류를 어떻게 찾아내고 고칠 것인가?

d. 멀티 코어 프로세서에서 스레드 응용 프로그램 성능을 어떻게 향상시킬 수 있고, 추가 코어로 어떻게 성능을 개선할 수 있는가?

인텔 패러렐 스튜디오는 위의 문제들을 해결해 줍니다.

아래 목록은 인텔 패러렐 스튜디오 도구가 어떻게 위의 문제들을 해결하는지를 간략히 보여줍니다.

인텔® 패러렐 컴포저	인텔 C++ 컴파일러와 포괄적인 스레드 라이브러리 및 디버거 익스텐션은 Microsoft Visual Studio 개발 환경에서 스레드 C/C++ 응용 프로그램을 빠르게 생성하고 디버깅할 수 있게 지원합니다. 이 도구를 사용하면 응용 프로그램에 가장 적절한 병렬 프로그래밍 모델을 선택할 수 있습니다.	문제 B 및 C 해결
인텔® 패러렐 인스펙터	멀티 스레드 응용 프로그램 개발을 위해 고안된 이 도구는 디버깅, 테스팅 및 검증을 더욱 쉽게 만들어 멀티 스레드로의 전환을 용이하게 합니다.	문제 C 해결
인텔® 패러렐 앰플리파이어	병렬 응용 프로그램이 멀티 코어에서 성능을 최적화할 수 있도록 성능을 분석하고 도구를 조정합니다.	문제 A 및 D 해결

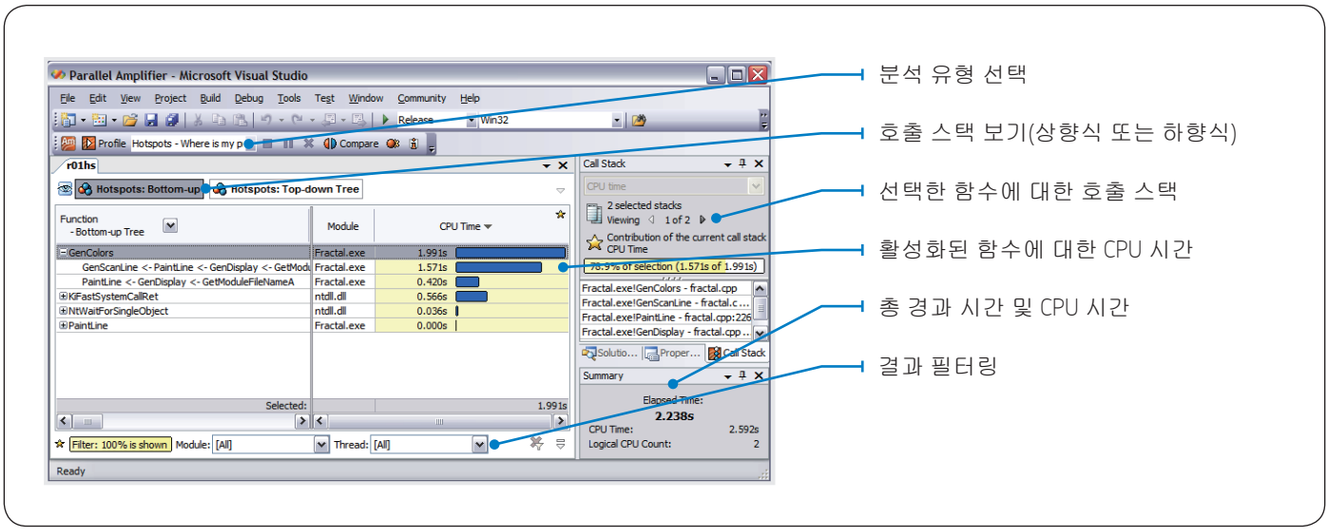

그림 2. 핫스팟 분석: 응용 프로그램이 어디에서 시간을 소비하는가?

먼저 응용 프로그램에서 가장 많은 시간을 소비하는 함수를 찾습니다. 프로그램을 더욱 빠르게 만들기 위해 병렬성을 조정하거나 추가할 수 있는 곳이 이 함수입니다. 또한 인텔 패러렐 앰플리파이어가 스택을 표시하기 때문에 함수가 어떻게 호출되는지를 알 수 있습니다. 호출 시퀀스가 여러 개인 함수의 경우 호출 스택 중 하나가 다른 스택보다 더 문제가 되는지 알 수 있습니다.

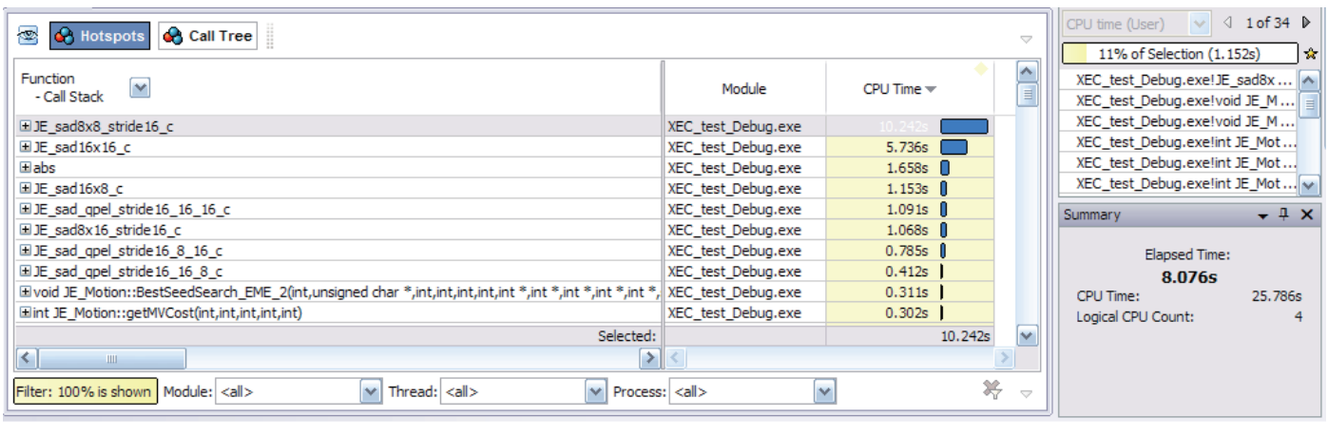

그림 3. 핫스팟 분석 보기를 통해 CPU 시간이 가장 높은 함수 파악

다음 작업 흐름 도표는 모든 인텔 패러렐 스튜디오 도구에 해당하는 일반적인 사용 모델을 설명합니다. 이제 막 응용 프로그램에 병렬성을 추가하기 시작했다면 핫스팟을 찾는 것이 첫 번째 단계일 것입니다. 이미 병렬성을 일부 추가했거나 응용 프로그램이 최적화되었다면 코드에 오류가 없다는 사실을 검증하거나 튜닝을 함으로써 시작할 수 있을 것입니다.

인텔 C++ 컴파일러:
Microsoft Visual Studio통합, Microsoft Visual C++ 호환 및 다양한 병렬 프로그래밍 API(Application Programming Interface) 지원

인텔 패러렐 스튜디오의 모든 기능은 Microsoft Visual Studio 2005* 및 2008에 통합됩니다.

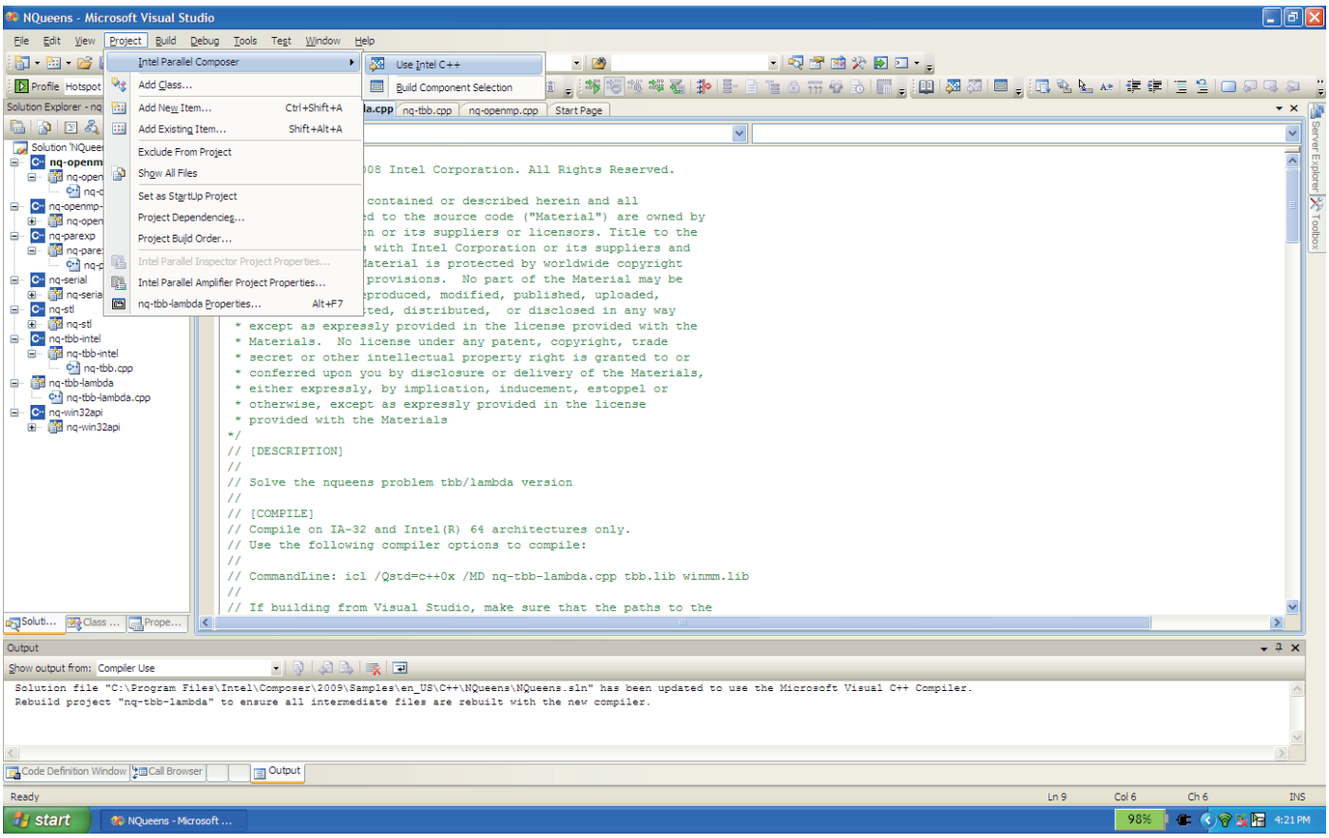

그림 4. 인텔® 패러렐 컴포저는 Visual Studio*에 통합됩니다. 화면의 솔루션은 인텔® C++ 컴파일러로의 전환 방법을 보여줍니다. 프로젝트(Project) 메뉴를 통해 또는 솔루션이나 프로젝트 이름 위에서 마우스 오른쪽 단추를 눌러 Visual C++로 쉽게 전환할 수 있습니다

인텔 스레딩 빌딩 블록: 병렬성 개선을 위한 C++ 라이브러리

인텔 스레딩 빌딩 블록(인텔 TBB)은 C++ 프로그램에서 병렬성을 표현하기 위한 다양한 방법을 제시합니다. 인텔 패러렐 스튜디오에 포함된 인텔 TBB는 인텔 C++ 컴파일러 또는 Microsoft Visual C++와 함께 사용될 수 있습니다. 인텔 TBB는 성능과 확장성을 위해 플랫폼 및 스레딩 정보를 추출하는 더 높은 수준의 태스크 기반 병렬성을 이용하는 라이브러리입니다. 인텔 TBB는 런타임 기반 프로그래밍 모델을 사용하고 표준 템플릿 라이브러리(STL)와 유사한 템플릿 라이브러리를 기반으로 병렬 알고리즘을 제공합니다.

인텔 TBB 태스크 스케줄러는 사용자를 위해 로드 밸런싱을 수행합니다. 스레드 기반 프로그래밍 시 스스로 로드 밸런싱을 수행해야 할 때가 종종 있는데 결코 쉬운 일은 아닙니다. 인텔 TBB 스케줄러는 프로그램을 여러 작은 태스크로 나눈 후 최상의 확장성을 구현할 수 있도록 태스크를 스레드에 고르게 할당합니다. 인텔 TBB는 C++에서 이미 사용 중인 방법을 확장해 병렬성 개념을 구현합니다. 인텔 TBB는 parallel_for, parallel_ while, parallel_reduce, pipeline, parallel_sort, parallel_scan 등 일부 함수와 템플릿을 컨커런트 컨테이너(concurrent container)와 함께 제공해 코드에서 병렬성을 개발할 때 생산성을 향상시킵니다.

```
void SerialApplyFoo( float a[], size_t n ) {
 for( size_t i=0; i!=n; ++i )
  Foo(a[i]);
}
```

그림 5A. 직렬 예제

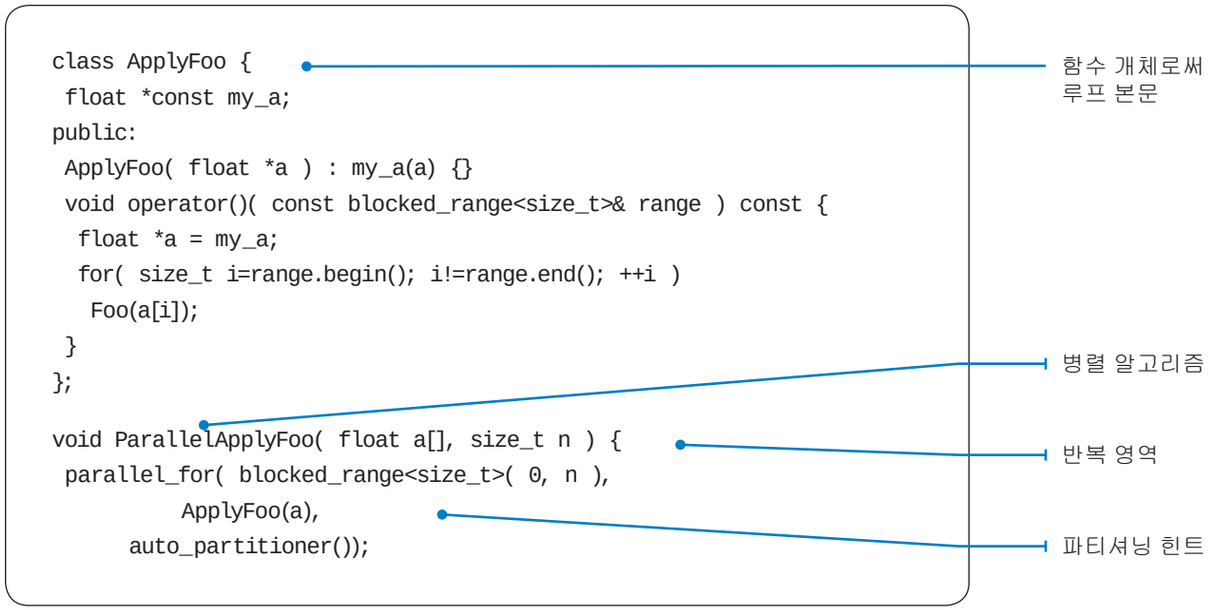

그림 5B. 인텔® TBB의 병렬 버전

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. 예를 들어, 루프에서 반복자를 사용하는 C++ 응용 프로그램이 있다면, 인텔 TBB는 람다와 함께 템플릿 루프 패턴의 구현을 지원합니다.

그림 6의 소스 코드는 람다 식으로 만든 함수 개체의 예입니다. C++와 인텔 TBB를 더욱 밀접하게 통합하면 람다 함수와 클로저를 사용해 코드를 매개변수로 넘김으로써 functor operator() 개념을 간소화할 수 있습니다.

```
void ParallelApplyFoo(float a[], size_t n ) {
  parallel_for( blocked_range<size_t>( 0, n ),
    [=](const blocked_range<size_t>& range) {
    for( int i= range.begin(); i!=range.end();
++i )
      Foo(a[i]);
    },
    auto_partitioner() );
}
```

그림 6. 람다 함수의 소스 코드 예

OpenMP 3.0*

OpenMP는 이식 가능한 멀티 스레드 응용 프로그램 개발의 업계 표준입니다. OpenMP는 파인-그레인(fine-grain: 루프 레벨) 및 코스-

그레인(coarse-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 않은 직렬 실행 코드와 동일합니다. 이를 통해 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* 3.0 태스크 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 2.5로 가능한 것 이상으로 불규칙한 병렬성을 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 7은 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 7. OpenMP 3.0 태스크 큐잉의 예

그림 7의 예에서는 단 하나의 태스크 큐가 필요합니다. 따라서, 단 하나의 스레드(omp single)를 사용해 큐를 설정해야 합니다. setQueen 호출은 서로 독립적이기 때문에 태스크 개념에 잘 들어맞습니다. 또한 전용 창의 OpenMP 프로그램에서 tasks, teams, locks, barriers, taskwaits의 상태 검사를 쉽게 만드는 인텔® 패러렐 디버거 익스텐션에 대해 읽어보는 것도 좋습니다.

단순 병행 함수

인텔 패러렐 컴포저는 네 개의 새로운 키워드를 제공해 OpenMP를 통한 병렬 프로그래밍을 더욱 쉽게 만듭니다. 새로운 키워드는 __taskcomplete, __task, __par, __critical입니다. 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 방식에는 아무 변화없이 계산이 병렬화됩니다. 그림 8은 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 8. 인텔® 패러렐 스튜디오의 인텔® C++ 컴파일러에 새롭게 추가된 4개의 단순 병행 함수 중 하나인 __par의 예

인텔® 통합 성능 프리미티브(인텔 IPP)

인텔 패러렐 컴포저는 멀티미디어, 데이터 처리, 통신 응용 프로그램을 위해 멀티 코어에 최적화된 소프트웨어 함수의 광범위한 라이브러리인 인텔 IPP를 포함합니다. 인텔 IPP는 비디오 코딩, 신호 처리, 오디오 코딩, 이미지 처리, 음성 코딩, JPEG 코딩, 음성 인식, 컴퓨터 영상, 데이터 압축, 이미지 색상 변환, 크립토그라피/CAVP 검증, 스트링 처리/규칙적 식, 벡터/매트릭스 수학 등에서 자주 사용되는 기본 알고리즘을 포함하는 최적화된 수천 개의 함수를 제공합니다.

인텔 IPP 함수는 스레드에 안전하며 대다수가 내부적으로 스레딩되어 있어 현재의 멀티 코어 프로세서로부터 최대의 혜택을 얻을 수 있습니다.

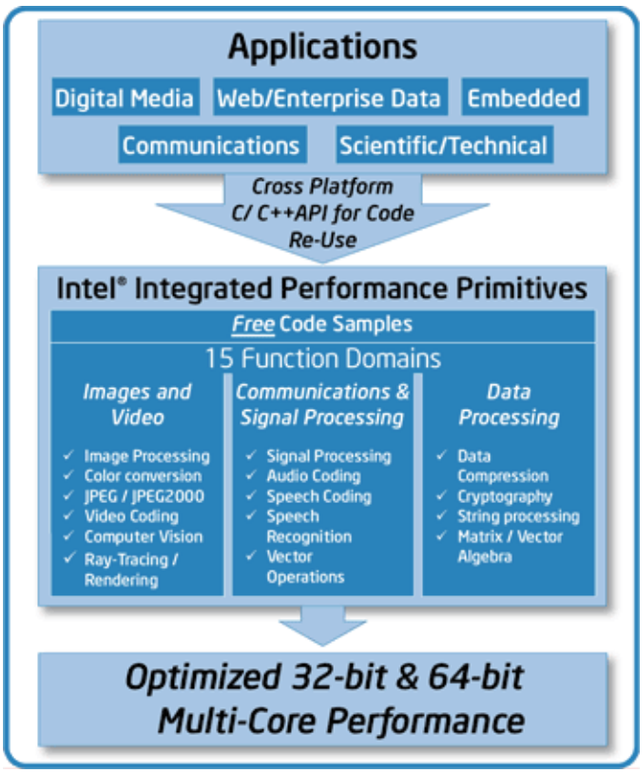

그림 9. 인텔® 통합 성능 프리미티브는 인텔® 패러렐 스튜디오의 일부인 인텔® 패러렐 컴포저에 포함되어 있으며, 다양한 도메인상에서 스레딩되고 스레드에 안전한 라이브러리 함수를 제공합니다

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 경우로써 암시적 valarray 루프가 자동 백터화되거나 인텔 성능 프리미티브의 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 10을 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 10. 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 공통 라인 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.

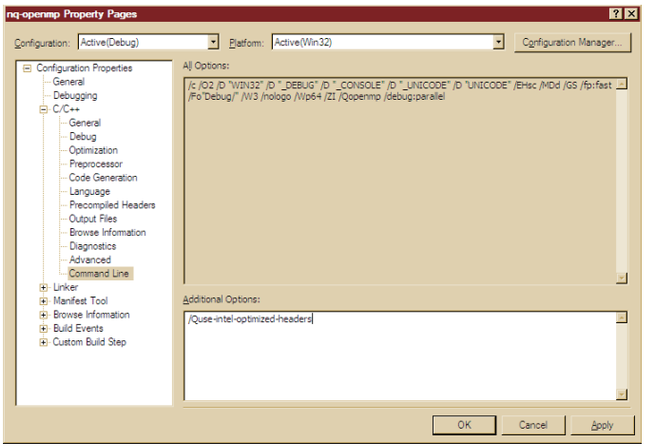

그림 11. 최적화된 헤더 파일의 사용을 위해 명령어를 Visual C++*의 명령 라인에 추가하기

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 12는 이를 보여주는 예입니다. 그러고 나면 응용 프로그램을 재구축할 수 있고, 응용 프로그램을 변경할 때와 마찬가지로 성능과 행동을 검사할 수 있습니다.

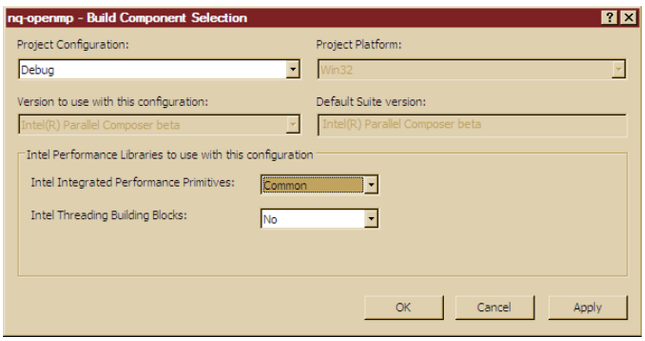

그림 12. Visual Studio*에 인텔® IPP를 사용하라고 지시

인텔® 패러렐 디버거 익스텐션

인텔 패러렐 컴포저는 설치 후 Visual Studio의 Debug(디버그) 풀 다운 메뉴를 통해 액세스할 수 있는 인텔® 패러렐 디버거 익스텐션을 포함합니다(그림 13 참조).

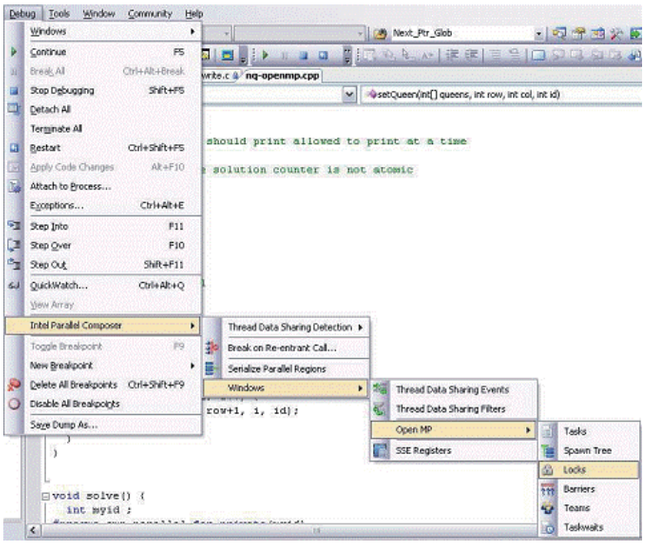

그림 13. 인텔® 패러렐 디버거 익스텐션은 Microsoft Visual Studio*의 디버그(Debug) 풀 다운 메뉴를 통해 액세스가

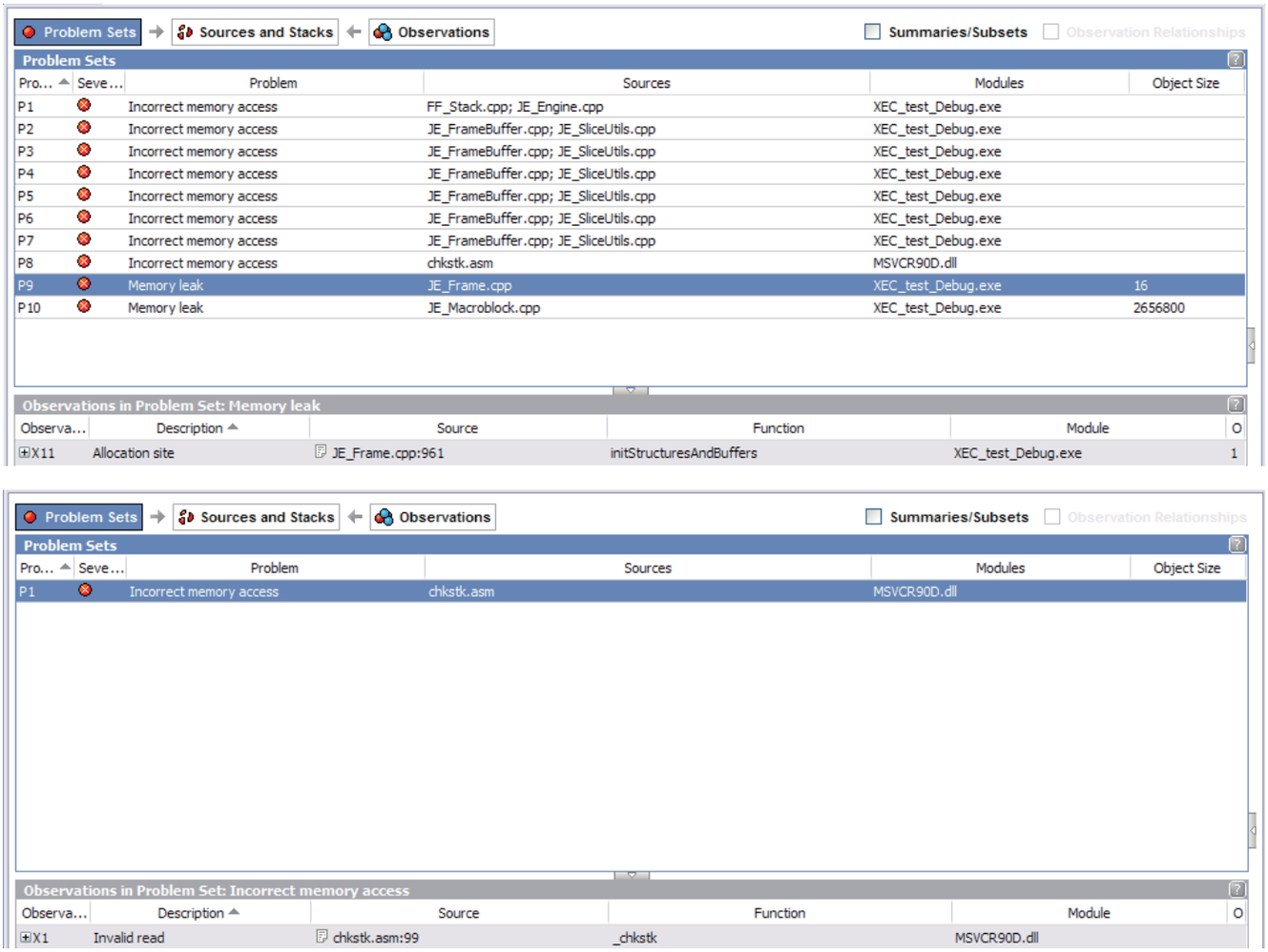

그림 14. 인텔® 패러렐 인스펙터의 수정 전과 수정 후 화면
(잘못된 메모리 액세스와 누수 수정)

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 정보는 http://software.intel.com/en-us/articles/parallel-debugger-extension/의 "인텔® 패러렐 디버거 익스텐션" 백서를 참조하십시오. 이 백서는 인텔 디버거 익스텐션에 대한 더욱 상세한 정보와 디버거 익스텐션이 제공하는 혜택, 디버거 익스텐션의 활용 방법 등을 제공합니다.

인텔 패러렐 인스펙터

스레딩 및 메모리 오류 발견

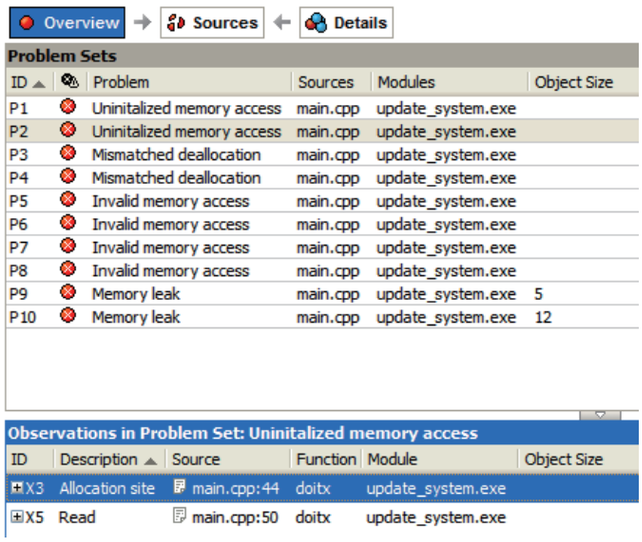

그림 15. 메모리 오류 발견

단일 및 멀티 스레드 응용 프로그램에서 누수 및 손상 등 메모리 오류를 신속히 찾아냅니다. 응용 프로그램 출시 전에 메모리 오류를 찾아내므로 지원 비용을 절감할 수 있습니다.

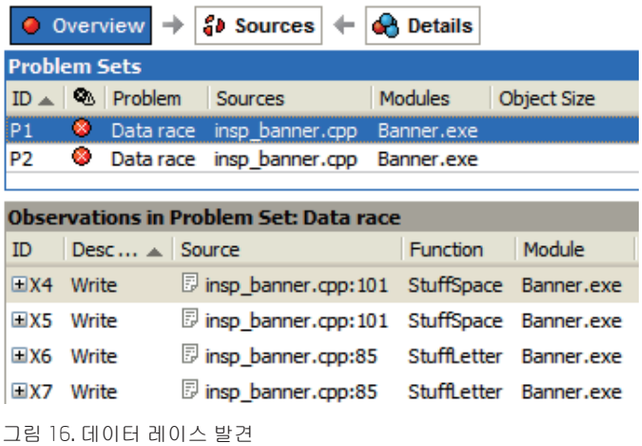

그림 16. 데이터 레이스 발견

데드락, 데이터 레이스 등 잠재적인 스레딩 오류를 정확하게 찾아내 디버거를 비롯한 기타 도구로 발견되지 않는 공통 오류로 인한 중단 및 충돌을 감소시킵니다.

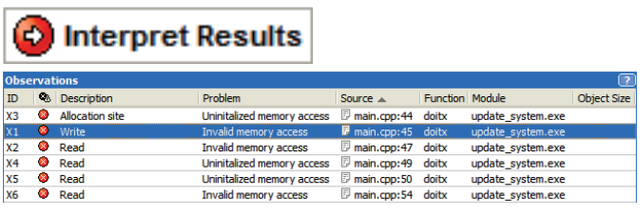

그림 17. 직관적인 그룹화

서로 연관된 문제를 그룹화해 개발자들에게 직관적인 인터페이스를 제공합니다. 한 문제를 수정하면 인텔 패러렐 인스펙터가 해당 수정 사항이 적용되야할 모든 위치를 알려줍니다.

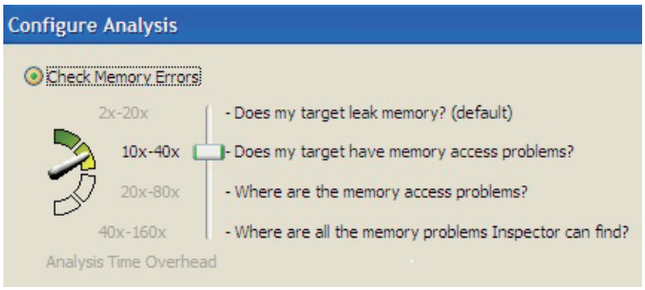

그림 18. 분석 수준 설정

간단한 분석 설정을 통해 개발자들은 실행 시간 대비 분석 수준을 제어할 수 있습니다.

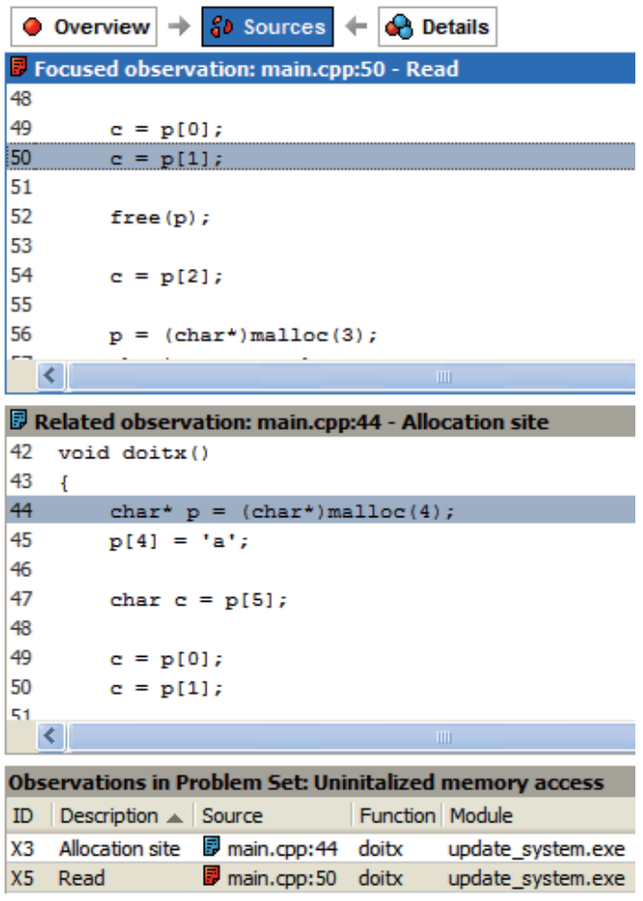

그림 19. 소스 코드와 매핑된 에러 식별

식별된 문제를 클릭하면 소스 코드가 나타나고 문제 코드에 바로 접근할 수 있어 신속한 수정이 가능합니다.

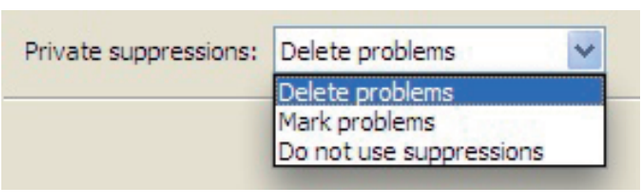

그림 20. 결과 삭제

관계없는 결과를 삭제해 분석 대상 정보를 줄여줍니다.

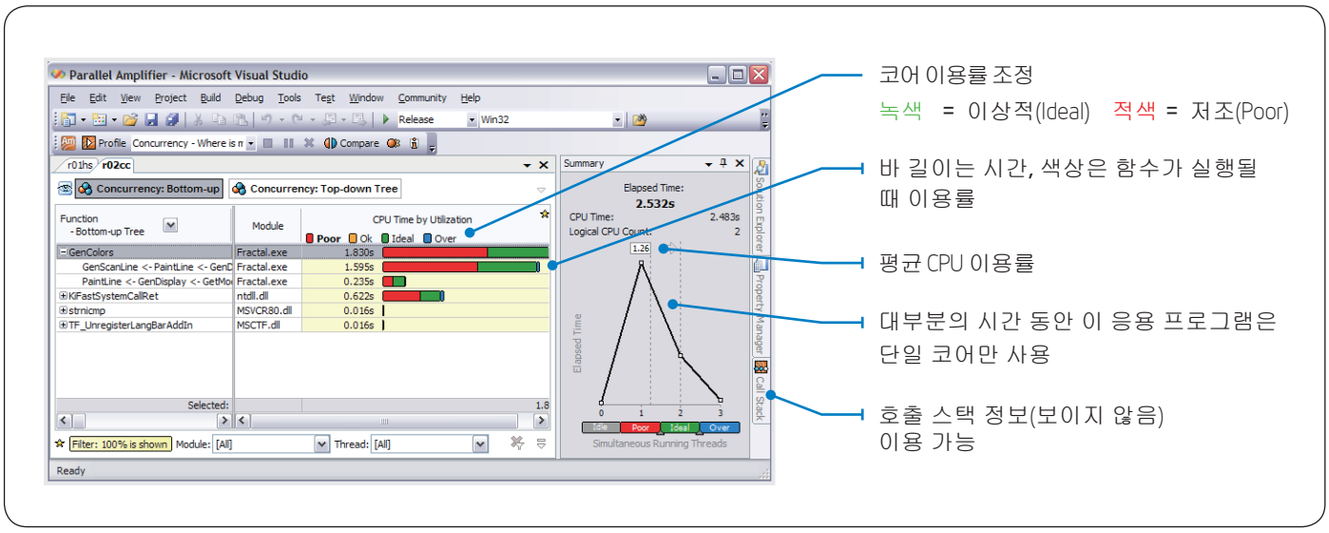

그림 21. 병행성 분석: 언제 코어가 유휴 상태에 있는가?

핫스팟 분석과 마찬가지로 병행성 분석은 가장 많은 시간을 소비하는 함수를 찾아냅니다. 또한 멀티 코어를 얼마나 잘 활용하고 있는지도 보여줍니다. 색상은 함수 실행 중 코어 이용률을 가리킵니다. 녹색 바는 모든 코어가 작동 중이라는 의미입니다. 적색 바는 코어가 충분이 이용되고 있지 않다는 것을 의미합니다. 적색이 있을 경우 병렬성을 추가해 모든 코어가 작동하게 합니다. 이를 통해 코어가 추가되면서 응용 프로그램 성능이 개선됩니다.

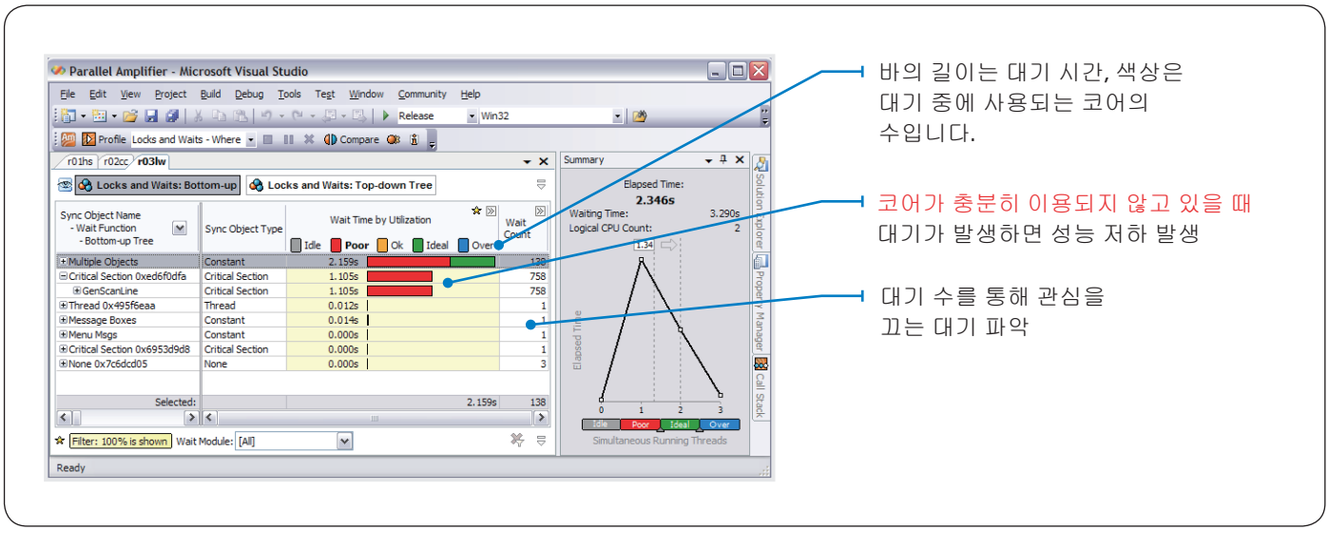

그림 22. 잠금 및 대기 분석: 불필요한 대기가 발생하는 곳은 어디인가?

잠금 상태에서 지나치게 긴 대기는 성능 문제의 공통적인 원인입니다. 코어가 모두 사용중(busy)일 때 대기하는 것은 나쁘지 않습니다(녹색). 사용되지 않는 코어가 있을 때 대기하는 것은 좋지 않습니다(적색).

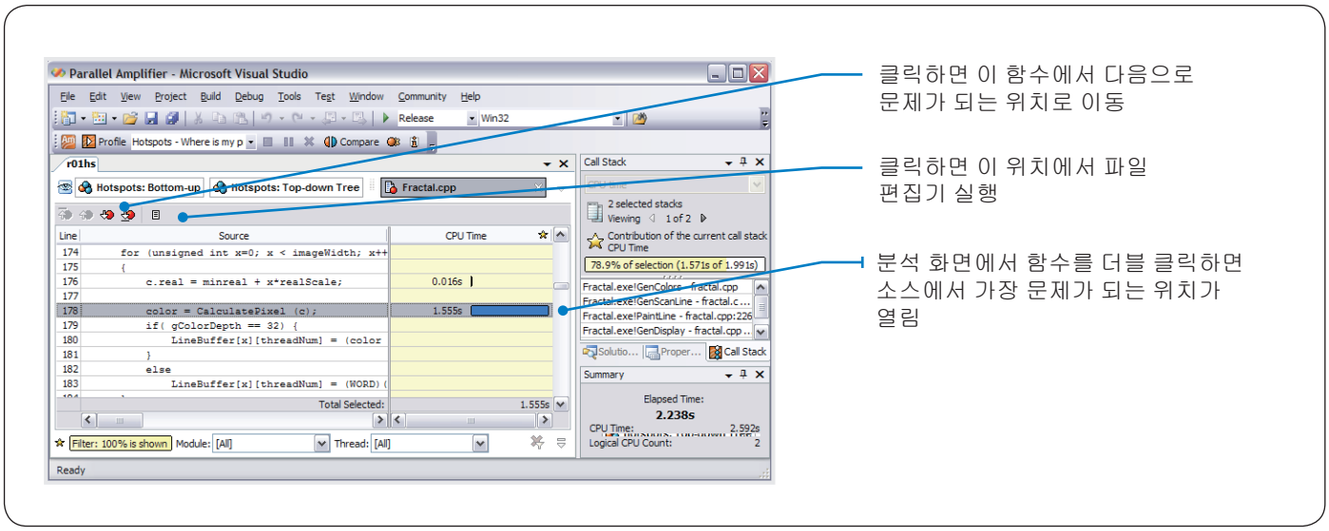

그림 23. 소스 보기: 소스에서 결과 보기

소스 보기는 소스 상에서 정확한 위치를 보여줍니다. 분석 화면에서 함수 이름을 더블 클릭하면 소스를 볼 수 있습니다.

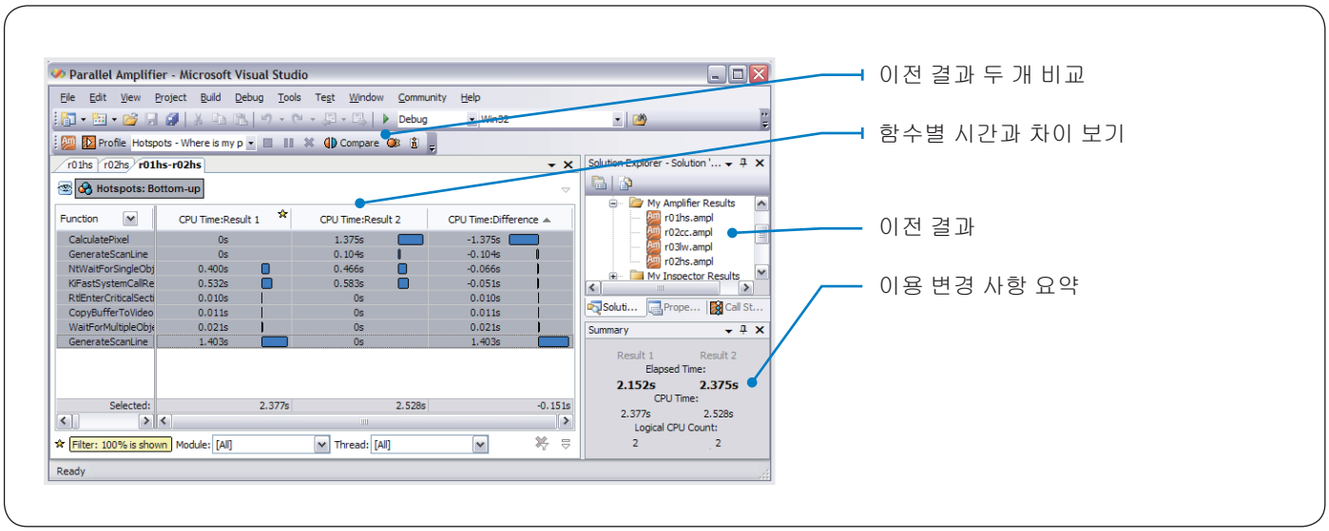

그림 24. 결과 비교: 변경 사항의 빠른 파악

튜닝 시 진행 상황을 빠르게 검사할 수 있고 성능 저하를 간편하게 분석할 수 있습니다.

특징

현재의 응용 프로그램 및 미래의 혁신적인 소프트웨어 개발자를 위해 설계되었습니다. Visual Studio*에서 병렬 프로그램을 설계 및 구축하는데 필요한 모든 것을 제공합니다.

- Microsoft Visual Studio에 완벽히 통합
- 최신 OpenMP* 사양 지원
- 기존 직렬 응용 프로그램과 새로운 병렬 --응용 프로그램이 멀티 코어를 활용하도록 준비하고 다중 코어(manycore)를 위한 "포워드 스케일(forward scale)"에 대비할 수 있도록 지원
  - 소스 코드에 대한 투자와 개발 환경 보존
  - 빠르게 성장하는 설치 기반 멀티 코어 시스템 활용
- 인텔 패러렐 스튜디오 도구는 설계 지원, 코딩 및 디버깅, 코드 내 오류 유무 검증, 튜닝에 이르는 개발 수명 주기의 각 단계에서 이용 가능
- 내장 지침 및 권장 포함, 수천 개의 코드 옵션을 포함하는 스레딩 라이브러리에 대한 액세스 제공
- 스레딩 간소화, 버그 및 시스템 성능 문제 감소 능력 제공, 기능이 풍부하고 향상된 제품을 더 빨리 출시할 수 있게 지원
- 설계자와 개발자를 위한 설계, 수십 년간 축적된 인텔의 소프트웨어 개발 제품 제공 경험을 바탕으로 고성능 컴퓨팅(HPC)을 위한 기술 응용 프로그램, 데이터베이스, 계산 집약 응용 프로그램, 스레드 응용 프로그램 실행 지원

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 웹 사이트를 참조하십시오:

http://software.intel.com/sites/support/

베타 버전 출시

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

0209/BLA/CMD/PDF 321552-001