Adding Parallelism with Intel® Parallel Advisor: No Parallelism Experience Required

Mark Davis, Intel
August 16th, 2010

mark dot davis at intel.com
Intel Parallel Advisor?

- A product in the upcoming release of Intel® Parallel Studio, which is a plug-in for Microsoft* Visual Studio
- A design tool that assists in making good decisions to transform a serial algorithm to use multi-core hardware
- A serial modeling tool that uses annotations in the serial code to calculate what might happen if that code were to execute in parallel as specified by the annotations
- A methodology and workflow to educate users on an effective method of using parallel programming
Digression – Intel® Parallel Studio

- Decide where to add the parallelism
  - Analyze the serial program
  - Prepare it for parallelism
  - Test the preparations
- Add the parallelism
  - Intel® Cilk™ Plus, Intel® Threading Building Blocks, ...
- Find logic problems
  - Only fails sometimes
  - Place of failure changes
- Find performance problems
Workflow

- Transforming many serial algorithms into parallel form takes 5 easy high-level steps
- Often existing algorithms are over-constrained by serial language semantics, and the underlying mathematics has a natural parallel expression *if you can just find it* (Advisor toolbar)
Overview

- If you look at these steps in more detail, you find decisions you will need to make.
- You do not have to choose the perfect answer the first time, so you can go back and modify your choices.
Serial Modeling (of Parallelism)

- Your application can’t fail due to bugs caused by incorrect parallel execution (it's running serially)
- You can evaluate several different proposals before committing to a specific implementation
- You can refactor your code to prepare it to be easier to incorporate parallel frameworks later
- Many transformations that make your code easier to parallelize, also make it easier to read and maintain
- All of your test suites should still pass when validating the correctness of your transformations
Programming Model (review)

**Fork-Join Parallelism:**

- Master thread spawns a **team of threads** as needed
- Parallelism is added incrementally: that is, the sequential program evolves into a parallel program

![Diagram of Master Thread and Parallel Regions]

**Advisor Serial Modeling**
Amdahl's Law

- (paraphrased) “The benefit from parallelism is limited by the computation which remains serial”
- If you perfectly execute ½ of your application in parallel you will achieve < 2x speedup
- The implication of this is that you must focus your attention where your application spends its time
- Running a little used section of your application in parallel just because it is easy gives you no benefit
Survey

Find the places that are important to your application
Annotation Concepts

- Advisor uses 3 primary concepts to create a model
  - SITE
    - A region of code in your application you want to transform into parallel code
  - TASK
    - The region of code in a SITE that you want to execute in parallel with the rest of the code in the SITE
  - LOCK
    - Mark regions of code in a TASK that must be serialized

- All of these regions may be nested
- You may create more than one SITE
Propose how you would like to partition your algorithm
Annotations - LOCK

Each annotation in a lock annotation pair acts as an executable statement that either acquires the lock or releases it. Locks must be released exactly once after they are acquired. If necessary, adjust the placement of the lock annotations to ensure this.
Suitability - Data Collection

Analyze your proposal to see if you made a suitable choice
Suitability

See if your SITE meets your performance expectations

Par Lab Boot Camp
Suitability - Improvements

Try alternatives to see how much they may improve your model

Par Lab Boot Camp
Correctness – Data Collection

Analyze your annotations to see if you made a correct choice.
Correctness

You may have Data Communication or Memory Reuse problems to fix.
Correctness - Problems

Each problem shows how the observations in each TASK relate
Repeat...

- You do not have to choose the perfect answer the first time, so you can go back and modify your choices

- Iterative refinement will either
  - Create a suitable and correct annotation proposal
  - Conclude no viable sites are possible

- Efficiently arriving at either answer is valuable
The Summary viewpoint gives you a high-level overview of your annotations and analysis results.

After fixing any problems you will have an annotated parallel model which is suitable and correct.

Next you choose a Parallel Framework, and replace the annotations with executable code.
Parallel Framework – TBB Loops

Counted Loops - Intel TBB

When tasks are loop iterations, and the iterations are over a range of values that are known before the loop starts, the loop is easily expressed in Intel TBB.

Consider the following serial code and the need to add parallelism to this loop:

```
ANNOTATE_SITE_BEGIN(sitename);
for (int i = lo; i < hi; ++i) {
    ANNOTATE_TASK_BEGIN(taskname);
    statement;
    ANNOTATE_TASK_END(taskname);
}
ANNOTATE_SITE_END(sitename);
```

Here is the serial example converted to use Intel TBB, after you remove the Advisor annotations:

```c++
#include <tbb/tbb.h>
...
    tbb::parallel_for( lo, hi,
        [&] (int i) { statement; }
    );
```

The first two parameters are the loop bounds. As is typical in STL programming, the lower bound is inclusive and the upper bound is exclusive. The third parameter is the loop body, wrapped in a lambda expression. The loop body will be called in parallel by threads created by Intel TBB. As described before in Create the Tasks, Using C++ structs Instead of Lambda Expressions, the lambda expressions can be replaced with instances of explicitly defined class objects.

**Parent topic:** Adding Intel TBB Code to Create the Parallel Tasks
Parallel Framework – TBB Tasks

Create the Tasks - Intel TBB
The following sections describe various alternatives, depending on how the tasks fit within the surrounding parallel site.

Two or More Parallel Statements
When the outermost statements in the annotation site have been placed into tasks, as shown in this serial example, it is easy to execute them in parallel.

```cpp
ANNOTATE_SITE_BEGIN(sitename);
ANNOTATE_TASK_BEGIN(task1);
  statement-1
ANNOTATE_TASK_END(task1);
ANNOTATE_TASK_BEGIN(task2);
  statement-2
ANNOTATE_TASK_END(task2);
ANNOTATE_SITE_END(sitename);
```

Two or More Parallel Statements - Intel TBB
In Intel TBB 3.0, the easiest way to cause several sequential statements to be executed as independent tasks is to change your program as follows using parallel_invoke.

Both of the following examples use the C++0x lambda expression feature – you need to use the Intel Parallel Composer compiler and enable the C++0x support to compile it, as described in Using the C++0x Lambda Expression Support described below.

```cpp
#include <tbb/tbb.h>
...
tbb::parallel_invoke(
  [=]{statement-1;},
  [=]{statement-2;}
)
```

A variable used inside a lambda expression but declared outside it is said to be captured. The ` [=]` in the example specifies capture by reference. Hence, if any variables are shared by the two statements, and one of the statements modifies the variable, then the variables should be protected by locking. It is also possible to capture by value ` [=]`, or even capture different variables different ways. See the documentation on lambda expressions for details.
Parallel Framework – (more)

Using C++ structs Instead of Lambda Expressions
Any code that can be written with a lambda expression can be written without one - it is just more work. All a lambda expression does is:

1. Define a class with operator() defined to execute the body of the lambda expression.
2. Define a class constructor that captures variables into fields of the class.
3. Construct an instance of that class.

The constructor can capture any of the surrounding locals that are needed and save them in data members.

```cpp
{ struct S1 { void operator()() { statement-1 }};
 struct S2 { void operator()() { statement-2 }};
 tbb::parallel_invoke(S1(),S2());
}
```

Parent topic: Adding Intel TBB Code to Create the Parallel Tasks

See Also
Overall Workflow Diagram
Recap – Intel Parallel Advisor

original serial C/C++ program
Recap - Intel Parallel Advisor

Run Survey tool to measure where your application is spending time

- original serial C/C++ program
- Advisor Survey tool
Recap - Intel Parallel Advisor

Run Survey tool to measure where your application is spending time

Add annotations to describe how you might divide the work of the hot-spot into parallel tasks
Recap – Intel Parallel Advisor

The annotated program still has serial semantics. Working on it is just normal programming.
Recap - Intel Parallel Advisor

Use the performance modeling (suitability) and race detection (Correctness) tools to identify potential issues.

The annotated program still has serial semantics. Working on it is just normal programming.
Recap - Intel Parallel Advisor

Use the performance modeling (suitability) and race detection (Correctness) tools to identify potential issues.

The annotated program still has serial semantics. Working on it is just normal programming.

Adjust the annotations or the program code itself to resolve performance and correctness issues.
Recap - Intel Parallel Advisor

The final version of the annotated program is projected to have good performance and no races.
Recap - Intel Parallel Advisor

Now, it's time to change the annotations into real parallel code – TBB or Intel® Cilk™ Plus or your favorite model. This straightforward translation is done by the programmer.
Recap - Intel Parallel Advisor

Use the other tools in Intel Parallel Studio to work with your parallel application (compile, debug, tune, etc)

parallel C/C++ program

other

Intel Parallel Studio tools
Conclusion

- The Intel Parallel Advisor is a unique tool
  - assists you to work smarter though detailed modeling
  - guides you through the necessary steps
  - leaves you in full control of your code and architectural choices
  - lets you transform serial algorithms into parallel form faster

- The serial modeling methodology
  - maintains your original application’s semantics and behavior
  - helps find the natural opportunities to exploit parallel execution
Intel® Parallel Studio

- More information about the Parallel Studio is available online, including a 30-day free trial


- Parallel Advisor, and Support for Visual Studio 2010 currently in Beta test.
- Send me email to try Advisor: mark dot davis at intel.com
Questions...