Intel® Cilk™ Plus Tutorial Instructions

Workshop on Programming of Heterogeneous Systems in Physics
Jena, 5-7 October 2011

1 Overview

This tutorial aims to familiarize you with Intel® Cilk™ Plus. Intel Cilk Plus is a C/C++ language extension providing fork-join task-parallelism using a work-stealing task scheduler, implicit auto-vectorization as well as data-parallel language constructs explicitly aiming for vectorization (array notation).

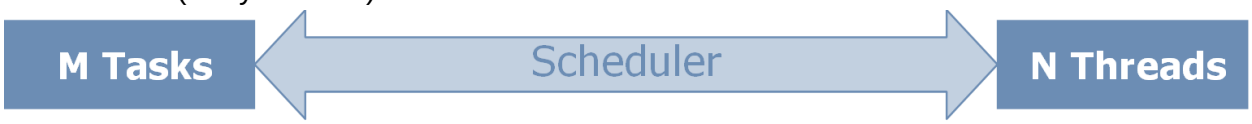

Figure 1: Preferring tasks over manual thread handling aims for performance portability rather than accidently tuning for the developer's environment.

Cilk is easy to start with due to the nature of a language extension. Cilk can be combined for example with Intel® Threading Building Blocks (Intel® TBB) which provides a rich and versatile set of parallel algorithms (patterns), and generic concurrent data structures (collections).

Through this tutorial, a general matrix-matrix multiplication ("gemm" with α = 1, and β = 0) will be developed using Cilk `for` as well as the Cilk array notation. Cilk reducers can be used, or reductions over array sections can be employed. As time permits, the performance can be analyzed using the Intel® VTune™ Amplifier XE.

2 Benchmark

The tutorial can be run as a competition in case the tutorial class wants to see benchmark results. Groups or individuals may submit their achieved implementations. Correctness and performance can be evaluated, and compared against other implementations or against baselines of interest. Please also take a note of the Optimization Notice. Moreover, the results should not be used to make performance conclusions in general, i.e. fixed function libraries are likely able to better exploit the hardware potential. In terms of the programming model, a single algorithm or implementation is unlikely able to represent a programming language's capabilities.

3 Tutorial

Source code files are numbered according to the exercise, i.e. "`00`" (usually Getting Started), "`01`" and so on. The names of files (base name) being part of a suggested solution are post-fixed with "`_solution`". Solutions may be distributed later during the tutorial, or they are given by the next exercise (eventually including bonus parts). Answers to questions are generally not part of the tutorial solutions. Take your own notes, and keep your own solutions!

The build system is based on GNU Make which can be also used on Windows* (via MinGW/msys); for convenience Microsoft* Visual Studio projects and a solution file are provided as well. To build or run an exercise, e.g. type:

```
make NAME=00_getting_started
make NAME=00_getting_started ARGS="1 5 1981" run
```

Please refer to the file `BUILD.txt` to learn more about the Makefile-based build system.

Other useful resources:
http://software.intel.com/en-us/articles/intel-c-composer-xe-documentation/
http://software.intel.com/en-us/articles/intel-cilk-plus-support/
http://software.intel.com/en-us/forums/intel-cilk-plus/
http://software.intel.com/en-us/articles/intel-learning-lab/ (video tutorials etc.)

Getting Started

Familiarize with the given code, employ Cilk `for`, and learn how easy it is to combine Intel Cilk and Intel TBB. In the bonus part, learn about utilities such a function to measure the execution time.

- Have a look at the given source code `00_getting_started.cpp`. Ask questions and make sure to understand the driver code in general.
- Turn the `for` statement(s) in `sgemm` into `cilk_for` without introducing data races!

Bonus

I. Run `00_getting_started` (no `cilk_for`), and write the duration down.
II. Rework `sgemm` to exactly implement the interface as shown below.

```
template<typename T>
void gemm(T* result, const T* a, const T* b,
          std::size_t arows, std::size_t acols,
          std::size_t bcols);
```

III. Note the `cilk/cilk.h` header in the given code, try `_Cilk_for` instead of `cilk_for`.
IV. What kind of analysis when using Intel® Inspector XE would be used to find data races?

Reducers

Gain a sense for typical data races and learn about Cilk reducers.

- Identify the variable which needs to become a reducer in order to turn the remaining `for` statement(s) into `cilk_for`. Have a look at the following example:

```
cilk::reducer_opadd<int> r;
```

- Remember the terms "initial view" and "identity". In order to keep the code similar to the initially given code, what needs to be employed – an initial view or an identity? Adjust the code accordingly!

Bonus

I. Run your `00_getting_started` (incl. `cilk_for`), run your `01_reducer` (incl. reducer), and write both durations down. Try the environment variable `CILK_NWORKERS`, repeat the experiments, and write the additional two durations down using only a single worker thread.
II. Consult the documentation to get an idea on how a reducer in C looks like.

Array Sections

As an alternative to the Cilk reducers as employed in the last exercise, the Cilk array notation (sometimes called "C Extended Array Notation") offers a built-in syntax for data-parallelism using the C array type (similar to the Fortran array section syntax).

Figure 2: In contrast to auto-vectorization (e.g. applicable to `for` and `cilk_for`), an explicit way to generate vectorized code is given by built-in vectors (Cilk array notation). In contrast to user-mandated vectorization (`#pragma simd`), correct code is generated in any case rather than mandating SIMD code even against the compiler's heuristics and analysis.

- Modify `gemm` in `02_cean.cpp` to use `__sec_reduce_add` to express what is implemented there using the Cilk reducer. The operand of this reduction operator is a multiplication of two array sections. Note that one of the sections is always a column of matrix `b` (stride).
- Is the new code subject of data races? Look at operators such as `+=`, or `+`.

Bonus

I. Make a table of results collected so far (eventually track results for `CILK_NWORKERS=1` as well).
II. Run your `02_cean` (incl. `__sec_reduce_add`), and compare against previous implementations.

Analysis

Performance analysis allows finding and qualifying problems, and to get rid of guess-work thus spending time effectively.

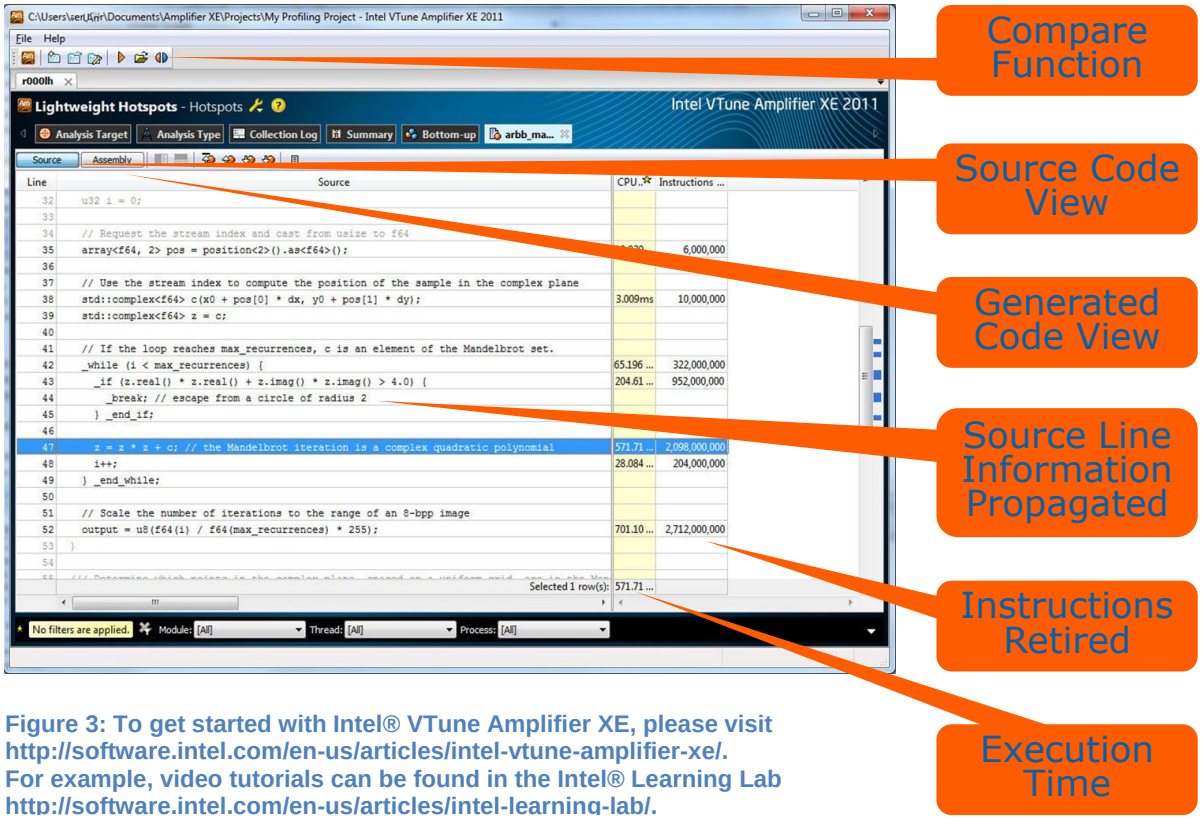

Figure 3: To get started with Intel® VTune Amplifier XE, please visit http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/. For example, video tutorials can be found in the Intel® Learning Lab http://software.intel.com/en-us/articles/intel-learning-lab/.

- Compare the implementations of `gemm` in `02_cean.cpp` and `03_gemm.cpp`. What is the difference with respect to the memory access pattern?
- Why is `cilk_for` not used for the inner loop of `gemm` in `03_gemm.cpp`?
- Build `02_cean_solution.cpp` (or your `02_cean.cpp`) using debug symbols.
  `make NAME=02_cean_solution OPT=Og` (Linux)
  `make NAME=02_cean_solution OPT=Od DBG=1` (Microsoft* Windows)
  Use the stand-alone GUI of Intel VTune Analyzer XE, create a project and point to the executable (eventually set the environment variable `CILK_NWORKERS=1`). Create a new analysis ("General Exploration"), and start the analysis.
- Build `03_gemm.cpp` using debug symbols, e.g. type:
  `make NAME=03_gemm OPT=Og` (Linux)
  `make NAME=03_gemm OPT=Od DBG=1` (Microsoft* Windows)
  Use the stand-alone GUI of Intel VTune Analyzer XE, create a project and point to the executable (eventually set the environment variable `CILK_NWORKERS=1`). Create a new analysis ("General Exploration"), and start the analysis.
- Use the comparison function of Intel VTune Analyzer XE to find differences between the two result data collections gathered before.

4 Disclaimer and Legal Information

5 Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804