An Integrated Tools Platform for Multi-Core Enablement

Beth Tibbitts
IBM T.J. Watson Research Center
tibbitts@us.ibm.com

Evelyn Duesterwald
IBM T.J. Watson Research Center
duester@us.ibm.com

Abstract

The growing number and scale of available multi-core systems changes the landscape of parallel computing. Parallel computing is no longer the domain of a relatively small group of highly skilled parallel programmers. In addition, programming communities that traditionally are not accustomed to reasoning about concurrency are suddenly faced with the challenge to exploit the potential of the new parallel hardware. We believe that a key component in the enablement of multi-core systems is the provision of advanced and highly integrated tools that service the needs of both the expert and the novice parallel programmer. This paper describes the Parallel Tools Platform (PTP) for OpenMP, as an attempt to provide such an integrated tools environment for multi-core enablement. Our main contribution in PTP is a suite of Parallel Language Development Tools that cover a spectrum of development needs ranging from mere convenience to automatic error detection.

1. Introduction

Parallel programming has traditionally been the domain of a relatively small group of expert High Performance Computing (HPC) programmers. The significant challenges involved in effectively programming parallel machines have prompted the HPC community to develop a variety of sophisticated parallel tools such as parallel debuggers, race detection and performance tuning tools. Unfortunately, these tools are often difficult to use, support only a few platforms, and may not be compatible with each other. As a result, is it not uncommon to find HPC programmers working with a set of stand-alone command-line tools and Emacs-style editors. Meanwhile, major leaps have been made in the development environments of other programming communities, namely, the commercial programming community which typically uses object-oriented languages, such as Java. Today's commercial programmers are accustomed to working in advanced highly integrated development environments that offer capabilities that range from mere convenience support to incremental compilation and automatic error detection.

The disparity among development environments has not been of much concern in the past as the two communities have had few interactions. However, the advent of multi-core computing is poised to change that by significantly altering the landscape of parallel programming. Multi-core computing opens up the domain of parallel programming to new communities, which raises a number of important questions. How can we enable a novice parallel programmer, who is not used to reasoning about parallelism and to whom performance may not have been much more than an afterthought, to effectively utilize the new multi-core systems? And how can we bridge the disparity among the communities that are coming to parallel programming with different backgrounds and expectations?

We believe that the answer to both questions lies in the ability to provide a unifying integration platform for parallel tools that can service a range of communities. A common tools platform can integrate advanced productivity-oriented development tools with the "bare-metal" performance-oriented tools that are familiar to the HPC community. The novice parallel programmer is more likely to exploit the performance potential of the new multi-core systems if the task of performance tuning is seamlessly integrated in the familiar environment of advanced development tools. The HPC programmer benefits by leveraging the advances in programmer productivity from modern integrated tool environments while cooperating with the usual HPC tooling capabilities.

We are developing the Eclipse Parallel Tools Platform (PTP) [1] to provide such an environment. PTP is a multi-organizational open-source development effort that was established in 2005 by Los Alamos National Laboratories with the goal of producing an open-source industrial-strength platform that provides a highly integrated environment specifically designed for parallel application

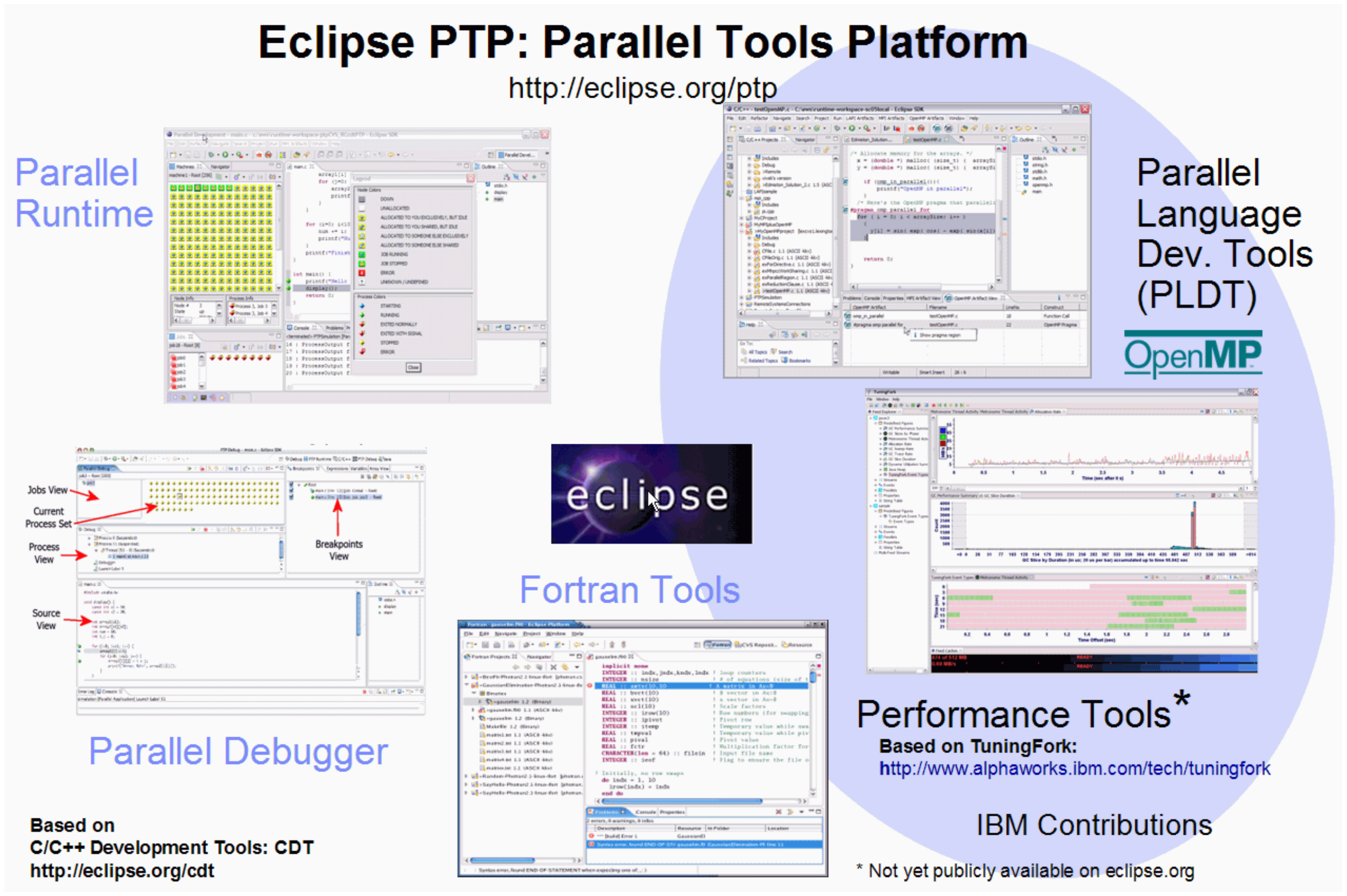


Figure 1: Overview of PTP with parallel runtime, parallel debugger, parallel language development tools, performance tools and Fortran tools.

development. PTP provides a unifying platform to integrate tools for all aspects of parallel program development, and a tools base upon which subsequent research into parallel tools can be done.

The current tools in PTP include a parallel runtime, a parallel debugger, parallel language development tools, and performance tools, as shown in Figure 1. PTP is a work in progress and there are ongoing efforts to support additional platforms and integrate additional tools. In this paper we focus on the IBM contributions to PTP, shown in the shaded circle in Figure 1.

2. PTP Overview

PTP is built on top of the Eclipse platform [2]. Eclipse is a platform that was designed to enable the integration of a wide range of programming and application development tools into a portable, scalable and open-source integrated development environment. Eclipse significantly reduces the cost of integrating tools by providing a single integration point that allows tools to integrate with the platform rather than each other. Eclipse supplies a large number of services, APIs and frameworks that can be leveraged by and shared among the tools.

PTP provides a set of tools that specifically tailor the Eclipse platform capabilities to the needs of parallel program development. The parallel runtime and debugger are platform dependent tools and PTP currently includes implementations based on MPI to address the needs of the HPC community. By porting runtime and debugger to a multi-core environment we can extend PTP for the multi-core user. An Eclipse-based IDE for the IBM Cell Broadband engine is already available as part of the Cell SDK 2.0 [3] and we are working to integrate the Cell support in PTP.

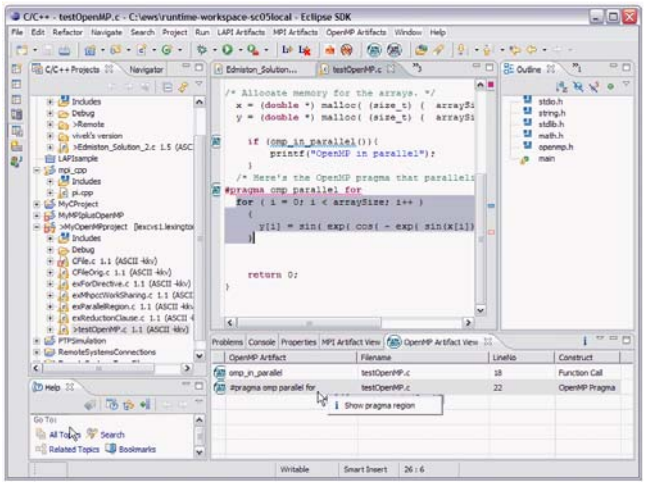
3. Parallel Language Development Tools

PTP's Parallel Language Development Tools (PLDT) provide tools to aid the parallel programmer during code creation or inspection. The PLDT is integrated by sharing an underlying layer of the Eclipse C/C++ Development Tools (CDT) [4]. All tools described in this section support C/C++. Some language tool functionality is also available for Fortran [5] and we are working to support Fortran in PLDT as well.

PLDT contains a spectrum of tools ranging from simple syntactical tools to sophisticated static analysis tools.

3.1 Assistance Tools

The PLDT assistance tools provide a set of syntactical services to the programmer including content assist for automatic completion of MPI and OpenMP APIs, complete with appropriate arguments. MPI and OpenMP "artifacts" are easily located in a separate view and mapped to source code lines. Built-in help files are available for MPI and OpenMP APIs as well. The scope of OpenMP `#pragmas` is also easily highlighted as illustrated in Figure 2.


C/C++ - testOpenMP.c - C:\ews\runtime-workspace-sc05local - Eclipse SDK
File Edit Refactor Navigate Search Project Run LAPI Artifacts MPI Artifacts OpenMP Artifacts Window Help
C/C++ Projects
Navigator
Edmiston_Solution...
testOpenMP.c
Outline
Problems Console Properties MPI Artifact View OpenMP Artifact View
Show pragma region
Writable
Smart Insert
26 : 6

```
/* Allocate memory for the arrays. */
  x = (double *) malloc( (size_t) (  arraySi
  y = (double *) malloc( (size_t) (  arraySi

    if (omp_in_parallel()){
        printf("OpenMP in parallel");
    }
  /* Here's the OpenMP pragma that paralleli
#pragma omp parallel for
  for ( i = 0; i < arraySize; i++ )
    {
      y[i] = sin( exp( cos( - exp( sin(x[i])
    }


    return 0;
}
```

OpenMP Artifact	Filename	LineNo	Construct
omp_in_parallel	testOpenMP.c	18	Function Call
#pragma omp parallel for	testOpenMP.c	22	OpenMP Pragma

Figure 2: View of OpenMP assistance tools showing OpenMP artifacts, `#pragma` region locator, etc.

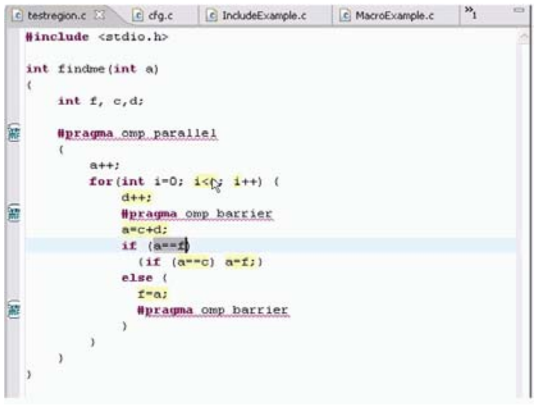
The PLDT tools also include project wizards to construct a managed build environment for parallel applications.

3.2 OpenMP Usage Checker

Along with the analysis for finding OpenMP artifacts, common problems are also located, and shown in the OpenMP Problems view. Like the OpenMP Artifacts view, the OpenMP Problems view can be used to navigate to the source code line by double-clicking on the line in the problems view.

The usage problems detected by the OpenMP usage checker include the following:

- Parallel directive dynamically inside another parallel directive
- For directive embedded within critical, ordered, or master extents
- For directive embedded within another parallel for or parallel sections
- For directive embedded within another for, sections, or single directive
- Barrier directive not permitted in region extent of for, ordered, sections, single, master, and critical
- Master directive not permitted in dynamic extent of for, sections, or single directives
- Ordered directive not permitted in dynamic extent of critical region



```
#include <stdio.h>

int findme(int a)
{
    int f, c,d;

    #pragma omp parallel
    {
        a++;
        for(int i=0; i<c; i++) {
            d++;
            #pragma omp barrier
            a=c+d;
            if (a==f)
              {if (a==c) a=f;}
            else {
              f=a;
              #pragma omp barrier
            }
        }
    }
}
```

Figure 3: Concurrency analysis view.

3.3 Concurrency Analysis

Built on top of the above analyses are more sophisticated static analyses, such as static concurrency analysis. Our OpenMP concurrency analysis is based on the algorithm developed by Yuan Lin [6]. Lin's analysis is a "nonconcurrency" analysis, that is, it answers the question whether two statements will not be executed concurrently by different threads. The analysis is conservative in that if the analysis determines two statements as non-concurrent they indeed cannot execute concurrently. However, if the analysis cannot prove that two statements are nonconcurrent, they may, but need not, execute concurrently.

The analysis builds an OpenMP control flow graph and an OpenMP region tree of the OpenMP construct structure. Using these two graphs the analysis determines the static computation phases in the program imposed by synchronization, including explicit synchronization through barriers or implicit synchronization through the termination of a parallel construct. Pairs of nonconcurrent statements are then determined based on conditions and relationships among the static phases that they belong to.

Figure 3 shows a view of the OpenMP concurrency analysis in PTP. The user can select any expression in the OpenMP code and the analysis will, in response to a context menu action, highlight all expressions that may execute concurrently.

4. Other PLDT Tools

We are working on expanding our repertoire of PLDT tools for OpenMP with a synchronization verification tool. We are porting the barrier matching analysis [7] that been developed in PTP for MPI programs to OpenMP. The barrier analysis determines barrier synchronization errors and, if no such errors exist, matches the textual barrier statements that synchronize together. PLDT's barrier matching analysis was first developed for MPI programs since MPI does not place any constraints on the placement of barriers. OpenMP has some constraints on barrier placement. However, barriers may still be textually unaligned, making it difficult for the program to verify correctness.

5. Performance Tools

Another class of tools in PTP is performance analysis and visualization tools. We are integrating TuningFork [8], a trace visualization tool that is a freely available Eclipse application, initially developed for visualizing real-time data [9]. Figure 1 includes a screen shot of TuningFork. We are extending TuningFork to visualize concurrency and synchronization in OpenMP. In addition to performance navigation support we are working on TuningFork plug-ins for automatic performance bottleneck detection.

6. Conclusions

We described the Parallel Tools Platform for OpenMP that provides a highly integrated tools environment for multi-core systems. Our focus so far has been on the platform-independent Parallel Language Development Tools in PTP. We developed a range of development tools that address the needs of both novice and expert parallel programmers. PTP is an extensible open source Eclipse project and we expect that in the future, additional tools will be integrated and additional multi-core platforms will be supported through our contribution and through the contributions of others.

Acknowledgements

We thankfully acknowledge the work of Don Pazel, a former member of our group at IBM Research, for his initial implementation of the OpenMP concurrency analysis. This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002 as part of the IBM PERCS project.

References

[1] Parallel Tools Platform http://eclipse.org/ptp

[2] Eclipse IDE and tools platform http://eclipse.org

[3] Cell IDE: http://www.alphaworks.ibm.com/tech/cellide

[4] C/C++ Development Tools http://eclipse.org/cdt

[5] Fortran tools http://eclipse.org/photran

[6] Yuan Lin. Static Nonconcurrency analysis of OpenMP programs. In Proceedings of the First International Workshop on OpenMP, June 2005.

[7] Yuan Zhang and Evelyn Duesterwald. Barrier matching for programs with textually unaligned barriers. In Proceedings of the Symposium on Principles and Practice of Parallel Programming, March 2007.

[8] TuningFork: http://www.alphaworks.ibm.com/tech/tuningfork

[9] David F. Bacon, Perry Cheng, Daniel Frampton, David Grove, Matthias Hauswirth, and V.T. Rajan. On-line Visualization and Analysis of Real-time Systems with TuningFork. In Proceedings of the 15th International Conference on Compiler Construction, March 2006.