OpenTS: An Outline of Dynamic Parallelization Approach

Sergey Abramov[1], Alexei Adamovich[1], Alexander Inyukhin[2], Alexander Moskovsky[1], Vladimir Roganov[1], Elena Shevchuk[1], Yuri Shevchuk[1], and Alexander Vodomerov[2]

[1] Program System Institute, Russian Academy of Sciences, Pereslavl-Zalessky, 152020, Russia, Yaroslavl Region.
+7 08535 98 064 (phone&fax)
abram@botik.ru
[2] Moscow, 119192, Michurinsky prosp., 1, Institute of Mechanics of MSU, Russia

Abstract. The paper is dedicated to an open T-system (OpenTS) — a programming system that supports automatic parallelization of computations for high-performance and distributed applications. In this paper, we describe the system architecture and input programming language as well as system's distinctive features. The paper focuses on the achievements of the last two years of development, including support of distributed, meta-cluster computations.

1 Open T-System Outline

Open T-System (Open TS) is a recent dynamic program parallelization technology for high-performance and distributed applications. It originates from functional and meta-programming technologies [1, 2] and tries to achieve maximum performance of single/multi-processors, supercomputers, clusters and meta-clusters. Another goal was the development of easy-to-use tools for parallel programming, with high learning curve and easy legacy code support. With initial implementations of T-system dated back to nineties and end of eighties of the last century, Open TS is a third generation of the T-system [3]. The Open TS approach allows addressing in a uniform way parallel computing problem for mutli-core processors, SMP systems, computational clusters and distributed systems. As well, Open TS facilitates parallel applications with non-uniform parallelism grains or parallelism grains defined at runtime.

1.1 Related Work

The Open TS design utilizes many concepts of parallel computing. First of all, it devises high-level parallelizing approach, while many of them currently exist [4]. Secondly it utilizes an extension of C++ language to express parallelism, while many other extensions of C and C++ for parallel computing were developed [5]. Thirdly, the concept of T-system is based upon functional programming approach [1], that make it very similar to parallel implementations of functional languages [6]. At last, Open TS runtime implementation utilizes Distributed Shared Memory (DSM) [7], mutli-tier architecture [8] and C++ template- based design [9]. Here we note separately only small fraction of all works in this field, not comprehensive but representative, as we hope:

V. Malyshkin (Ed.): PaCT 2005, LNCS 3606, pp. 303 – 312, 2005.

1. Charm++ [10] is a C++ extension for parallel computing, which is used to create high-performance codes for supercomputers [11]. Open TS is different in many aspects – from runtime implementation to language semantics. The most important is that Open TS uses functional approach for parallelization, while Charm++ uses asynchronous communication with object-oriented model.
2. mpC++ is another example of successful implementation of "parallel C" for computational clusters and heterogeneous clusters [12]. While mpC uses explicit language constructions to express parallelism, Open TS has implicit parallelization constructs.
3. Cilk is a language for multithreaded parallel programming based on ANSI C [13]. Cilk is designed for shared memory computers only, in contrary Open TS can be run on computational clusters and meta-clusters.
4. Glasgow Parallel Haskell is a well-known extension of Haskell programming language [14]. Open TS is similar with GPH by utilizing some implicit approach to parallelizing computation, while enabling low-level optimization on C++ level, unavailable in Haskell.
5. OMPC++[15] is very similar to Open TS in many aspects, especially in the way of using C++ templates in runtime. However, language extensions are of primary importance for Open TS concept.

While many parallel programming techniques, like Unified Parallel C [16] and CxC [17] are not covered in our comparison, Open TS distinctions will be virtually the same.

1.2 Programming Model

Unlike many tools for parallel programming, T-System does not try to change the usual programming model too much. Native input language is a transparent attribute-based extension of C++; however, other T-dialects of programming languages are in the development stage: T-FORTRAN, T-REFAL. Only two new notions are really important for programming: T-function and T-value. T-values are extensions of basic C values with non-ready value, read access to a non-ready value stops execution of a T-function, unless C-value is provided during computation. T-functions are pure C-functions forming functional model at the top level of program structure. However, imperative C exists inside T-functions enabling potential for low-level optimization. Support for object oriented-model is forthcoming.

An important feature of Open TS is a separation of the computation code from the scheduling code. In Open TS, the programmer is enabled to develop complex strategies for dynamic parallelization without affecting the computational code itself.

1.3 Execution Model

Parallel execution is based on a completely conflict-free data-flow model, and the "macro-scheduling" algorithm distributes computational tasks (active T-functions) over all available computing resources on the fly. Thus, latency hiding should enable very high computational power utilization. Moreover, heterogeneous (e.g. different CPU speeds) computational clusters can be efficiently loaded with that approach.

Special hardware such as application-specific accelerators and processors can be also considered as specific computational resources, it is dynamically loaded in the same way.

Millions of threads[1] can work in a cooperative and conflict-free way enabling latency hiding: any time non-ready T-value is reached, T-System switches rapidly to another ready-to-compute task. In this way, T-System avoids blocking computation in many cases when communication infrastructure permits. In brief, T-System may be a good candidate to fill up the gap between fast recent CPUs and latency-restricted communications.

1.4 T-Applications

T-application is a self-contained, dynamically linked executable. In a nutshell, it recognizes the execution environment and automatically loads a corresponding communication driver on the fly. The execution environment may be one of the following.

- Unicomputer – runs as a single process
- SMP — runs on a machine with symmetric multi processing capabilities
- MPI (6 flavors are supported now, including PACX MPI and MPICH-G2 for the meta-cluster environment)
- PVM.

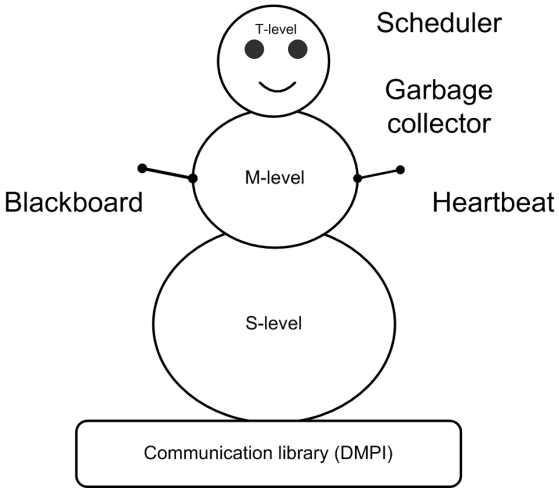
Thus, T-applications don't need to be recompiled or re-linked for all possible communication flavors. This is important in many cases, especially in meta-clusters with heterogeneous MPI implementations.

2 Open T-System Design Notes

Open T-System runtime has a microkernel-based design. Microkernel, or T-Superstructure, is a central part of the runtime. It contains all essential entities that a typical program needs to be run on. T-Superstructure has a “snowman” architecture of three tiers: `S' (“super-memory” and “super-threads”), `M' (mobile objects and references) and `T' (T-values, variables, references, functions). Being compact in size (less than 5 000 lines in about 100 C++ classes), it suits for various extensions: enhanced task schedulers, memory allocation schemes, custom thread systems, and so on. A special class 'Feature' is used to register extension plug-ins, which are typically dynamically linked at the startup stage. The microkernel can be easily ported to `almost pure' hardware, because it is almost self-contained. C++ [cross] compiler only is required for such porting. However, since C++ templates are used extensively, a modern C++ compiler is required.

Fast context switch is a special feature of Open TS, which is very important for efficient T-applications. Since T-applications are known to create millions of simultaneous threads, fast switching is key important. Today, the T-context switch is 10 times faster than the fastest standard thread library switch.

[1] Opens supports the usage of more than one million of threads even in one usual processor — this was shown practically, this was used in real applications.



A “Supermemory”, or special kind of distributed shared memory, is located outside of program data and used to manage T-values. Novel communication technologies such as hyper-transport can be directly incorporated into the “Supermemory” layer to avoid an unnecessary MPI overhead. Super memory is utilized in six different ways:

1. T-Values
2. Task exchange
3. Resource information exchange
4. Memorization table
5. “Heartbeat” (see below)
6. Shutdown signal

The fault-tolerance support has been implemented with the help of LAM MPI BLCR checkpoint system [18]. It is integrated with the T-System runtime, thus making fault-tolerant computing easier.

Since the T-system originates from the functional programming model, it is possible to implement the fault-tolerance on the base of re-computing of T-functions. This work is forthcoming.

3 Compilation of T-Programs

Two approaches are followed to develop compilers for T++ programs.

The first, “converter”, approach utilizes OpenC++ [19] parser to translate a T++ program to a C++ program using Open TS runtime library calls. Advantage of that approach is that the best-of-breed C++ compiler can be used, with the best processor-specific optimization available. The drawback is some C++ syntax features that are not supported seamlessly due to Open C++ limitations.

An alternate, “compiler”, approach is based on an open-source GNU C++ compiler. An extra front-end language for T++ has been implemented, it has a smooth and comprehensive support of all C++ language features. However, if the GNU C

compiler optimization is not on a par with the other compilers of the target platform, a performance loss might happen.

4 T-Application Development Stages

First of all, T++ is a transparent attribute-based dialect of C++. The T++ code can be trivially mapped to the sequential C++ program by masking T-attributes on the preprocessor stage. To start, the T++ code may be developed and debugged without T-System.

Then, the `t++' compiler may be used to obtain T-executables which should be able to normally run on the unicomputer. Thus, the second stage of the development process is to check whether everything works correctly on the unicomputer — this involves usual testing and debugging for the traditional (one-processor) case.

Furthermore, the same executable may be run on the `cluster emulation'. The simplest way to do this is to use LAM on various Linux systems: the command

```
mpirun n0,0,0,0 <t-executable>
```

will emulate the 4-node cluster. Some tuning can be done at this stage.

Finally, run T-executable on the desired target platform.

5 T-Application Debugging and Tuning

T-System has a number of built-in profiling, tracing and debugging facilities.

First of all, debugging is facilitated by several modes of compilation: “optimized”, “normal” and “debug”. The “optimized” mode uses the runtime version with heavy optimization. The runtime of the “normal” mode is simplified as compared to the “optimized” version. If an application is compiled in the “normal” mode and a problem persists, it should be attributed to the application itself — not the runtime — with high degree of confidence. Moreover, the “debug” mode generates a large amount of debug output, which helps programmers to understand the current situation in T-runtime and applications. This output can be filtered with the help of regular expressions.

A full-fledged Trace facility has also been implemented for T-applications.

When the program is finished, some statistical data is printed (see figures below). It includes minimal/medium/maximal (depending on computational nodes) values of the following parameters: used CPU time, communication time, idle time. This hot profiling information may be very useful for the tuning of applications.

Communication message logs can be called in order to understand which communication traffic occurred during the program execution. A T-function call graph can also be obtained.

If the program crashes, some information (including program call stack with source line numbers) is printed. Optionally, the debugger is started at the same time, which may be very convenient for a rapid problem discovery.

Finally, a special heartbeat logic is used to discover broken program/communication state. If heartbeat timeout is reached without any data exchange, then all T-processes will exit automatically.

6 Sample Program Run

The example program is the calculation the Fibonacci number. Since it is not very hard computationally, it is a good test for the runtime system, and it illustrates well the simplicity of T++ programming.

```
tfun int fib(int n)
{
          if (n<2) return 1;
          return (fib(n-1)+fib(n-2));
}
tfun int main (int argc, char *argv[])
{
          int n = atoi(argv[1]);
          printf("Fibonacci %d is %d\n",n,(int)fib(n));
          return 0;
}
```

The only T-function is the "fib" function which recursively calls itself. Since the result of "fib" is a non-ready value, explicit casting to **int** is necessary for the program to run correctly. The casting results in the "main" thread wait until the result of "fib" is ready. "fib" recursively calls itself creating a tree, while the tree branches can be computed in parallel.

Compiling the program is possible with either t++ or tg++.

```
t+ -o fib0 fib0.tcc
```

The process of the program execution is illustrated in Fig. 1 (running on single processor) and Fig. 2 (running on four-cluster nodes). You may see some speedup demonstrated by "fib". The example has been a mere illustration that doesn't reflect the real quality of T-system, benchmarking results will be published elsewhere.

```
[alexanderm@skif demos]$ ./fib 29
Open T-System Runtime v3.0, 2003-2004, PSI RAS, Russia.
Running under unicomputer MPI on 1-rank cluster:
  [3.3Gf,3322BM,0.86GiB]
Starting tfun main, good luck!

fib(29) = 514229
Tasks activated:     1664080
Tasks exported:      0
Msgs sent:           0
Async Msgs:          0
Msgs size:           0
Taskboard visits:    1664080
Scheduler time:      2.724
MPI time:            0.000
Idle time:           0.000
Tasks time:          30.260
Total time:          36.897
```

Fig. 1. Sample program run result in console

```
[alexanderm@skif demos]$ mpirun n1,2,3,4 ./fib 29
Open T-System Runtime v3.0, 2003-2004, PSI RAS, Russia.
Running under LAM MPI on 4-rank cluster:
  ([3.1Gf,3060BM,0.86GiB]+[3.1Gf,3060BM,0.49GiB]+[3.1Gf,3060BM,0.86GiB]*2) ~= [1
2.2Gf,12240BM,3.08GiB]
Starting tfun main, good luck!

fib(29) = 514229
Tasks activated:      [407582/416020/420993]
Tasks exported:       [33/35/40]
Msgs sent:            [1534/1577/1639]
Async Msgs:           [0/0/0]
Msgs size:            [114472/118405/124576]
Taskboard visits:     [408080/416657/421748]
Scheduler time:       [1.188/1.219/1.234]
MPI time:             [0.031/0.033/0.035]
Idle time:            [0.011/0.013/0.015]
Tasks time:           [8.942/9.059/9.190]
Total time:           [16.368/16.375/16.380]
```

Fig. 2. Sample program run on multiple cluster nodes

7 T++ Language in a Nutshell

The T++ language is a semantically and syntactically "seamless" extension of C++. The language constructions are enumerated below with short descriptions following them:

tfun — a function attribute which should be placed just before the function declaration. Now, the function cannot represent a class method but must be an ordinary C function. A function with the "tfun" attribute is named "T-function".

tval — a variable type attribute which enables variables to contain a non-ready value. The variable can be cast to the "original" C++-type variable, which makes the thread of execution suspend until the value becomes ready.

tptr — a T++ analogue of C++ pointers which can hold references to a non-ready value.

tout — a function parameter attribute used to specify parameters whose values are produced by the function. This is a T++ analog of the "by-reference" parameter passing in C++.

tct — an explicit T-context specification. This keyword is used for specification of additional attributes of T-entities.

tdrop — a T++ -specific macro which makes a variable value ready. It may be very helpful in optimization when it's necessary to make non-ready values ready before the producer function finishes.

8 Runtime Performance

The detailed performance study of Open TS runtime is out of the current paper scope and will be published elsewhere. However, overall runtime performance and quality is good enough to stimulate many groups outside of Program Systems Institute to

develop their own applications with Open TS (see below). Best speedup achieved with image-processing application is approximately 60% of linear speedup on 32-processor computational cluster with Scalable Coherent Interface (SCI) interconnect.

9 Applications

Approximately a dozen of applications have been developed with the help of T-system. Some of them are the following:

- Plasma physics modeling tool
- Aerodynamics simulation package
- Tools for computational modeling in chemistry
- Automatic text categorization package
- Radar image modeling application
- Remote sensing images processing

10 Support

Open T-System is being developed in the Program System Institute of the Russian Academy of Sciences (PSI RAS) as a key technology in the SKIF Super-Computing project. The system support can be obtained via e-mail: opents@botik.ru (developers' conference).

11 Work in Progress

We are also working on various application-oriented T-libraries. Such libraries are represented as the T++ code (working also in pure C++) and may be used without any knowledge of T++ or even parallel programming at all. Using the C++ inheritance mechanism, an application programmer just needs to define several application-specific methods — virtual functions — to obtain a complete highly-parallel computational component for a custom high-performance application. Other development areas if macro-scheduling schemas for meta-clusters and other distributed systems.

Acknowledgements

This work is supported by joint "SKIF" supercomputing project of Russia and Belarus and basic research grant from Russian Academy of Science program "High-performance computing systems on new principles of computational process organization" and basic research program of Presidium of Russian Academy of Science "Development of basics for implementation of distributed scientific informational-computational environment on GRID technologies".

References

1. Field A.J, Harrison P.: Functional Programming (International Computer Science Series), Addison-Wesley (1988)
2. (a) Turchin V.F.: The concept of a supercompiler Transactions on Programming Languages and Systems.—**v** .8, N 3 (1986) .292 –325. (b) Ershov A.P., D.,Futamura Yo.,Furukawa K.,Haraldsson A.,Scherlis W.L.:Selected Papers from the Workshop on Partial Evaluation and Mixed Computation. New Generation Computing. **v**.6 , N 2 –3.
3. (a) Abramov S. M., Adamovich A. I., Kovalenko M. R.: T-system as a programming environment with automatic dynamic support parallelization support. An example with implementation of ray-tracing algorithm Programmirovanie, № 25 (2), 100–107.(in Russian) (b) Abramov S. M., Vasenin V. A. , Mamchits E. E., Roganov V. A.: Slepukhin A.F. Dynamic parallelization of programs based on parallel graph reduction. A software architecture of new T-system version. Proceedings book of MIPHI scientific session, 22-26 January 2001, **v**. 2, Moscow, 2001. (in Russian)
4. High-level Parallel Programming and Applications Workshop 2003 Proceedings in Parallel Processing Letters , .**v**. 13, issue 3.
5. Gregory V.: Wilson (Editor), Paul Lu (Ed.) Parallel Programming Using C++ MIT Press, 1996
6. H-W. Loidl , F. Rubio , N. Scaife, K. Hammond , S. Horiguchi , U.Klusik , R. Loogen , G.J. Michaelson , R. Pena , S. Priebe ,A.J. Rebon and P.W. Trinder: Comparing parallel functional languages,: programming and performance. J. of Higher-order and Symbolic Computation, 2003
7. J.B. Carter, D. Khandekar, L. Kamb: Distributed shared memory: where we are and where we should be headed. Fifth Workshop on Hot Topics in Operating Systems (HotOS-V) May 04 - 05, 1995 Orcas Island, Washington
8. M. J. Vianna. E. Silva, S. Carvalho, J. Kapson,: In Proceedings of the 2nd European Conference on Pattern Languages of Programming (EuroPLoP '97). Siemens Technical Report 120/SW1/FB. Munich, Germany: 1997
9. Andrei Alexandrescu.: “Modern C++ Design: Generic Programming and Design Patterns Applied” , Addison Wesley Professional., ISBN: 0201704315;2001
10. L. V. Kaleev, Sanjeev, Krishnan :Charm++: Parallel Programming with Message-Driven Objects. In [5] 175-213
11. James C. Phillips Gengbin Zhengy Sameer Kumary Laxmikant V. Kaley :NAMD: Biomolecular Simulation on Thousands of Processors. In: Supercomputing 2002 conference proceedings http://sc-2002.org/paperpdfs/pap.pap277.pdf
12. Alexey Lastovetsky: mpC - a Multi-Paradigm Programming Language for Massively Parallel Computers, ACM SIGPLAN Notices, 31(2):13-20, February 1996
13. Cilk: Efficient Multithreaded Computing by Keith H. Randall. Ph. D. Thesis, MIT Department of Electrical Engineering and Computer Science. June 1998. http://supertech.lcs.mit.edu/cilk/
14. Pointon R.F. Trinder P.W. Loidl H-W.: The Design and Implementation of Glasgow distributed Haskell: IFL'00 - 12th International Workshop on the Implementation of Functional Languages, Aachen, Germany (September 2000) Springer Verlag LNCS 2011, pp 53-70
15. Yukihiko Sohda, Hirotaka Ogawa, Satoshi Matsuoka OMPC++ - A Portable High-Performance Implementation of DSM using OpenC++ Reflection. Lecture Notes In Computer Science; Vol. 1616 pp 215-234 Proceedings of the Second International Conference on Meta-Level Architectures and Reflection , 1999,

16. F. Cantonnet, T. El-Ghazawi: UPC Performance and Potential: A NPB Experimental Study, in Supercomputing 2002 conference proceedings http://sc-2002.org/paperpdfs/pap.pap316.pdf
17. CxC Programmer's Manual. Engineering Intelligence Corporation, 2004, available at http://www.engineeredintelligence.com/
18. Sankaran S. , Squyres J.M. , Barrett D., Lumsdaine A. , Duell J. , Hargrove P. Roman E. The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing, Proceedings, LACSI Symposium, October 2003, Sante Fe, New Mexico, USA.
19. Chiba S. A Metaobject Protocol for C++ , In: Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), page 285-299, October 1995.http://www.csg.is.titech.ac.jp/~chiba/openc++.html