Variable Reassignment in the T++ Parallel Programming Language

Alexander Moskovsky, Vladimir Roganov, Sergei Abramov, and Anton Kuznetsov

Program Systems Institute of the Russian Academy of Sciences,
Pereslavl-Zalessky, 152020, Yaroslavl region, Russia
moskov@lcc.chem.msu.ru,var@pereslavl.ru,
abram@botik.ru,tonic@pereslavl.ru
http://www.botik.ru/PSI

Abstract. The paper describes the OpenTS parallel programming system that provides the runtime environment for T++ language. T++ is an extension for C++ that adds a set of keywords to C++, allowing smooth transition from sequential to parallel applications. In this context the support of repeated assignments to a variable is an important feature. The paper focused on semantics and implementation of such variables in T++. Applications written in T++ can be run on computational clusters, SMPs and GRIDs, either in Linux or Windows OS.

Key words: OpenTS, T++, parallel computing, variable reassignment

1 Introduction

There are a lot of academic and industry projects exist in the field of high-level parallel programming, many of which are successful and well-known [1, 4, 2, 5, 3, 6–14, 17, 18]. However, none of them has widespread adoption outside high-performance computing community. Today, when parallel processors are more accessible to mass market than ever before (multi-core CPUs, small clusters), parallel programming tools has to be simple enough for mainstream use. That makes further research in the parallel programming field more necessary than ever before.

Our new and original OpenTS (Open T-system) approach has many advantages due to combination of simplicity and high performance. The main project goal is to ease the process of writing parallel programs for moderately experienced programmers. Therefore OpenTS users don't have to be able to write very efficient code, but nevertheless they can make quick though efficient parallel applications.

Three ideas are basic for the OpenTS:

- Parallel graph reduction is used as a programming model [11](coarse grain dataflow).
- Extending a sequential programming language (C,C++) with additional keywords to express parallelism.

– Dynamic parallelization at runtime on basis of directives (T++ attributes) specified by a programmer.

T++ is a native input language for OpenTS and is a transparent attribute-based extension of C++ [16] that supports parallelism. It is simple, intuitively obvious and easy to learn.

In this paper we focus on T++ language distinctive feature: ability to reassign variables. This feature differentiates our project from the other parallel programming systems based on functional programming (e.g. SISAL [14]).

The paper is organized as following. First, OpenTS programming model and language are described. Then, underlying shared memory mechanism is outlined. Finally, there are performance measurements and comparison of OpenTS application with an analogue.

2 OpenTS Programming Model

The OpenTS programming model is very similar to a coarse-grain dataflow model [10, 3]. In OpenTS in order to write a parallel program, a programmer must designate the following:

– Independent parts (pure functions) of the program (parallelism granules or T-functions), that can be moved over the network and computed remotely.
– Variables that are used to exchange data between parallelism grains (T-variables).

The rest of work is performed by the OpenTS components: T++ compiler and runtime support library.

It should be kept in mind that granules of parallelism must be large enough (coarse grains) to avoid overhead inflicted by the runtime system operating with relatively small grains. So, a grain with, say, a single floating-point multiplication would not be efficient, because time needed to transfer such grain to a computing node can be much greater than grain computing time. On the other hand, large grain size is often the cause of a small number of grains and unbalanced computational load in multiprocessor.

Granule aggregation technique is available in OpenTS for recursive programs, similar to "task inlining" technique of Multilisp [6]. In order to create an input programming language for OpenTS, existing programming language should be extended with extra keywords or pseudo-comments. Currently, only C++ extension "T++" is implemented, Refal [15] version is underway.

3 T++ Language

The T++ language adds the following keywords (attributes) to C++:

– **tfun** — a function attribute which designates a T-function that may return a non-ready value. T-function represents a granule of parallelism. As for now, a T-function cannot be a class method but must be an ordinary C function.

- **tval** — a variable attribute which enables variables to contain a non-ready value (T-value). Such values are produced by T-functions. At any moment of time T-values can be in one of two possible states: non-ready, when producer T-function is still working, or ready, when producer finished and returned a result. The T-variable can be cast to the original C++ type variable, what makes producer T-functions running and the thread of execution suspend until T-value becomes ready. That it very similar to "dataflow variable" [10] or "mentat variable" [3] or "futures" [6]. That also differs T++ from standard data-flow models, where task is ready for execution only after all incoming data is ready — in opposite, threads in OpenTS can be launched before any incoming data for a granule is ready.
- **tptr** — T-pointer, a T++ analogue of C++ pointer that can hold reference to a non-ready value (T-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 function that 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.
- **twait** — a T++-specific function that causes waiting for an argument expression to be ready.

Open C++ [21] is used for conversion of T++ programs into C++. It translates all T++ attributes into the pure C++ code.

The recursive calculation of the given Fibonacci number is the simplest parallel program:

```
#include <stdio.h>
tfun int fib(int n) {
    return (n < 2) ? n : fib(n-1) + fib(n-2);
}
tfun int main (int argc, char *argv[]) {
    if (argc != 2) {return 1;}
    int n = atoi(argv[1]);
    printf("Fibonacci %d is %d\n",n,(int)fib(n));
    return 0;
}
```

In this case, invocations of `fib` functions are treated as independent tasks that can be computed in parallel in independent threads, or on the remote computational nodes. You can see that minimal modifications differ the T++ from the C++ code: attributes of T-functions and explicit cast of `fib` function result to `int`. That casting not only extracts value from T-value, which is returned by `fib`, but also makes `main` function to wait for the `fib` result. However, runtime support library may implement a C-call for `fib`. In that case, overhead of calling

T-function drops dramatically, and parallelism granules (T-function calls) "aggregated" in a single granule, technique, similar of "task inlining" for MultiLisp [6].

Some specific of the T++ language should be underlined:

- It is a "seamless" C++ extension, which means that evident C++ macrodefinitions of T++ keywords can enable T++ program compilation by a C++ compiler. If some good coding style in T++ is adhered to, such compilation (which is done via "-not" command line option) will result in correct sequential program.
- Garbage collection is supported. Non-ready values that are no longer necessary are detected and destroyed by the runtime system.
- Function execution can be postponed, not necessarily generating any computation after invocation, depending on execution strategy. By default, if no thread is waiting for function result, function execution will be omitted.
- T-variables support repeated assignments. This is done by the tricky protocol of assignment and readiness of the variable values, related to the thread lifecycle.

4 Implementation of Variable Reassignment

The latter T++ feature deserves a more detailed description. Each T-variable is linked with its T-value. T-variables are type-safe: it is possible to assign values of the same type only. A T-variable can have multiple values during its lifetime. T-variables may share the same T-value. In other words, T-variables are wrappers for their values, however, variables may change their values in a way that C++ smart pointers [19] do. T-value can be either non-ready or an ordinary C-value. An assignment of a T-variable to a C-variable makes execution thread to wait until T-value is ready — a usual approach for "futures-based"[6] systems like OpenTS.

```
tval int x;
int y;
x = some_tfun();
y = x;  // will wait until x has a ready value
```

Contrary, assignment of a C-value to a T-variable immediately causes T-value of that variable to be ready.

```
tval int x;
x = some_tfun(); // assigns a non-ready value to x
x = 1; // assigns a ready value to x
x = 2; // unlinks the old value, creates a new one
```

The capability to assign multiple values for a single T-variable required us to introduce "producer" thread concept. For the sake of simplicity, let us consider each T-function call to be executed in a separate thread of execution (a "lightweight thread"). Then, we consider thread as a producer for all values that are allocated in its context. When producer thread is destroyed, all T-values produced by this thread are no longer changed (frozen). Consumer threads don't have any access to T-values, unless values are frozen. Frozen values are also produced when a T-function is called with T-variable as a parameter — snapshot copy of current value is produced. Such frozen values then can be easily shared among concurrent threads or across multiple cluster nodes. Consider the following example of T-variable reassignments:

```
#include <stdio.h>
#define N 10
tfun int tproducer(int i) {
    tval int x;
    x = 2*i;
    return x;
}
tfun int tconsumer(tval int t, int i) {
    return t+i;
}
tfun int main(int argc, char *argv[]) {
    tval int tmp;
    tval int res[N];
    for (int i = 0; i < N; ++i) {
        tmp = tproducer(i);
        res[i] = tconsumer(tmp,i);
    }
    for (int i = 0; i < N; ++i)
        printf("%d\n",(int)res[i]);
    return 0;
}
```

First, take a look at first loop inside `main` function. On each iteration, the `tmp` T-variable is assigned a new value — an output of `tproducer` for i-th iteration. On the next line, `tmp` is a parameter for `tconsumer` invocation: `tmp` value is passed as input to `tconsumer`. If `tmp` value was "hot" (like after `tmp=i` instead of `tmp=tproducer(i)` assignment), the value would be copied and the copy would be frozen. Inside the `tproducer`, the `x` value was initially allocated as "hot" and the `tproducer` is the value producer. Then the `x` variable is assigned with `2*i` value of type `int`. On the next line, the value of `x` variable is assigned to the return value of `tproducer` call. When the `tproducer` thread stops, the return value is "frozen" and delivered to consumers (to `tmp` and `t` variables). The `tconsumer` job is trivial — it awaits of its input value (produced by `tproducer`) and conducts a summation of two integers. On the next interation of the loop, reassignment to `tmp` will unlink `tmp`'s value of last iteration and the process will continue.

5 Distributed Shared Memory in OpenTS

The OpenTS implementation relies on object-oriented distributed shared memory (OODSM)[20] for storing T-values. The garbage collection is supported for T-values. When OpenTS runtime detects that there is no more links to a given T-value, the value is scrapped and memory address is reused. Each cell has "seqNo" attribute in order to distinguish between various "generations" of objects sharing the same cell.

OpenTS employs MPI[4] as a low-level transport layer. In this case, a "naive" reference count implementation of garbage collection is inapplicable. For instance, MPI library[4] can deliver a bunch of "decrement" messages ahead of corresponding "increments", which could result in premature value destruction. OpenTS utilizes a more sophisticated technique — weighted reference counting. In this approach, each reference has an integer "weight" depending on "weight" of value. A T-value is considered no longer necessary when its weight equals to the original weight assigned at value creation.

6 POV Ray Parallelization with OpenTS

There are a lot of applications that utilize OpenTS as a parallel programming platform. Most of them are simulation tools. Some are developed by groups outside of our institution, like [24, 25]. Here we present our case-study example: implementation of patch for POV Ray (Persistence Of Vision) ray-tracer. In order to evaluate the programming technique as a whole, not only the runtime support library effectiveness and scalability is an issue, but programming language qualities as well. Despite programming language beauty is a subjective matter, we believe that some sharp differences in code statistics, such as the number of lines of code, can be rather convincing.

The well-known POV Ray application is widely used to obtain realistic images using ray-tracing rendering technology. POV Ray is freely distributed with source code evolved from C to C++ during last years. Since ray-tracing consumes a lot of computation resources even for simple scenes, a few parallel versions of POV Ray have been developed and contributed by different authors. There are several approaches to parallelize POV Ray to make it work on multicomputers: from trivial rsh-based scripts, invoking POV Ray executable for parts of target scene on different UNIX hosts, to the most effective PVM and MPI-based implementations, supporting dynamic load balancing and features like animation and interactive display functions.

There are two well-known MPI-based POV Ray ports:

- MPI POVRay, based on POV Ray 3.1g., written in C with MPI patch applied.
- ParaPov, based on POV Ray 3.50c., written in C++ with MPI patch applied.

Total size of POV Ray 3.1g MPI-related source files (mpipov.c and mpipov.h) is more than 1500 lines of code, with multiple changes scattered over many files.

However, an intention to minimize changes in POV Ray code resulted in coding style that sometimes can be challenging to the reviewer. POV Ray 3.50c MPI patch is written in more straightforward C++, about 3000 lines total.

To make comparison more correct, we made our patch applicable to both original POV Ray versions (3.1g and 3.50c). OpenTS port is straightforward: most of porting work consisted in removing unnecessary task management MPI code, replacing it by only two T-functions. Result code is written in C/C++ with T++ patch applied, and no MPI code. T++ source file tpovray.tcc is shorter than 200 lines. Also a few minor changes were made in file povray.c.

Performance comparison has been done with the "chess board scene" taken from the original POV Ray distribution with the scene width and height set to 1500 pixels. The chess board scene has 430 primitives in objects and 41 in bounding shapes. The graph, displaying the ratio between execution times of MPI POV Ray 3.50c and OpenTS is shown on Fig. 1. In table 1 there are execution times for scalar, MPI and T++ versions of POV Ray.

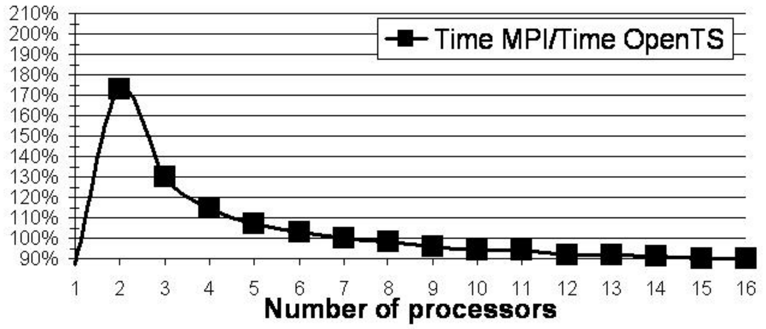

Fig. 1. Performance comparison of T++ and MPI versions of POV Ray

The computational cluster used had the following configuration:

- operating system: Red Hat Linux, kernel 2.4.27.
- 16 cluster nodes; each node: 2CPUs AMD Athlon MP 1800+ RAM 1GB, HDD 40GB.

The performance advantage of T++ version is due to suboptimal load balancing of MPI version. The latter reserves one CPU for management work, and advantage gradually degrades when number of CPUs increases.

7 Related Work

The comprehensive review of all research, conducted in the field of high-level parallel programming tools, extends far beyond limits of this paper. The review

N procs	C-POVRay	MPI-POVRay	T-POVRay
1	1 364.40	1 368.16	1 578.34
2		1 361.05	787.00
3		682.97	526.33
4		455.81	395.96
5		342.22	318.64
6		273.56	265.79
7		228.26	228.74
8		197.56	200.63
9		171.29	179.21
10		152.27	161.38
11		137.30	146.65
12		125.00	136.00
13		114.95	125.00
14		105.89	116.96
15		98.82	109.59
16		91.97	102.42

Table 1. Benchmark results (in seconds) for scalar, MPI and T++ versions of POV Ray

by D. Talia [17] which was written in 2000 has 62 citations in bibliography. Since then, interest to parallel programming tools only grew, since multi-core microprocessors appeared on commodity market and cluster computing became very popular.

We would like to stress distinctive features of OpenTS approach:

- OpenTS borrows many ideas from the world of parallel functional programming [8], that differentiates OpenTS from many parallel C++ extensions [18]. At the same time, OpenTS relies on C++ runtime, that overcomes performance limitations of functional programming languages.
- OpenTS has features like reassignment of variables and distributed garbage collector implemented. That differentiates OpenTS approach from the dataflow and future-based approaches like Mentat[3] or Oz[10].
- OpenTS adopts more higher-level implicit approach to parallelism than MPI [4].
- OpenTS has no means to parallelize computations in loops, like OpenMP [1], however, it is oriented primarily on computational clusters.

The way how values of variables in T++ become ready is similar to transaction concept of modern relational databases, however, OpenTS does not follow "transactional memory" approach to parallel programming [22, 23].

8 Conclusion

OpenTS is a tool for high-level parallel programming, providing a runtime for T++ language. It supports variable reassignment that helps in development of complex though efficient parallel applications. T-variable can be assigned a value multiple times, that conforms to a usual imperative style of programming. This feature considerably distinguishes OpenTS from many analogue parallel programming systems. As POV Ray case-study shows, only 200 lines of T++ code is required to parallelize it, while independently developed MPI version is more than 1500 lines long. At the same time, application performance is affected in a very little extent. Many aspects of the system are not covered in this paper, load balancing is the most important. We refer to our previous publication here [16]. The OpenTS approach to parallelism is implicit, since runtime library and compiler together should be able to adapt programs to a wide variety of parallel computers that exist today: multi-cores, SMPs, computational clusters with different kind of interconnects, and GRIDs. At the same time, computational source code of OpenTS is separated from management code (scheduling, task aggregation and so on). We hope that these features together will make OpenTS a useful tool for parallel computing. The OpenTS is available for download at `www.opents.net`.

Acknowledgments. This work is supported by Russian Foundation of Basic Research grant N 050708005ofi_a and basic research program of Presidium of the Russian Academy of Sciences “Development of basics for implementation of distributed scientific informational-computational environment on GRID technologies”.

As well, we thank Igor Zagorovsky, German Matveev, Alexandr Inyukhin, Alexandr Vodomerov, Eugene Stepanov, Ilya Konev, Elena Shevchuk, Yuri Shevchuk, Alexei Adamovich, Philip Koryaka, Maxim Kovalenko and others who contributed to the design and implementation of OpenTS and T++.

References

1. Chandra R., Menon R., Dagum L., Kohr D., Maydan D., Mcdonald J.: Parallel Programming in OpenMP Morgan Kaufmann 2000
2. Kaleev L. V. ,Krishnan S. Charm++: Parallel Programming with Message-Driven Objects in [18], 175-213.
3. Grimshaw A. S.: Easy to Use Object-Oriented Parallel Programming with Mentat IEEE Computer,May 1993, pp. 39–51
4. Lusk E. et. al. MPI-2: Extensions to the Message-Passing Interface MPI Forum, 2001
5. Randall K.H. Cilk: Efficient Multithreaded Computing, Ph. D. Thesis. MIT Department of Electrical Engineering and Computer Science. June 1998. `http://supertech.lcs.mit.edu/cilk/`
6. Halstead R,: MULTILISP: a language for concurrent symbolic computation ACM Transactions on Programming Languages and Systems (TOPLAS), **7**,4 (1985),501–538
7. Zhang L., Krintz C., Soman S.: Efficient Support of Fine-grained Futures in Java International Conference on Parallel and Distributed Computing Systems (PDCS), November 2006, Dallas, TX

8. 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**, 53–70
9. Lastovetsky A. mpC — a Multi-Paradigm Programming Language for Massively Parallel Computers ACM SIGPLAN Notices, February 1996, 31(**2**):13–20
10. Smolka G The Development of Oz and Mozart, Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers, ed. Peter Van Roy ,LNCS **3389**(2005), p 1
11. Loidl H-W.: Granularity in Large-Scale Parallel Functional Programming PhD. Thesis. University of Glasgow. March 1998. Available online: `http://www.dcs.gla.ac.uk/~hwloidl/publications/PhD.ps.gz`
12. Goodale T. et. al. The Cactus Framework and Toolkit: Design and Applications VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002., Selected Papers and Invited Talks, LNCS **2565**(2003) 197–227
13. Cantonnet F., El-Ghazawi T. Performance and Potential: A NPB Experimental Study Supercomputing Conference 2002, `http://sc-2002.org/paperpdfs/pap.pap316.pdf`
14. Cann D. Retire Fortran? Debate Rekindled. Supercomputing Conference, New-Mexico USA, November 1991.
15. Turchin V.F. REFAL-5 programming guide and reference manual New England Publishig Co., Holyoke 1989.
16. Abramov S., Adamovich A. I., Inyukhin A., Moskovsky A., Roganov V., Shevchuk E., Shevchuk Yu., Vodomerov A.: OpenTS: An Outline of Dynamic Parallelization Approach. Parallel Computing Technologies (PaCT)-2005, Krasnoyarsk, Russia, September 2005, LNCS **3606** (2005) 303-312.
17. Talia D.: Advances in Programming Languages for Parallel Computing in Annual Review of Scalable Computing (2000)//Yuen C. K. 28–58
18. Wilson G.V. (Editor), Lu P. (Editor) Parallel Programming Using C++ MIT Press, 1996
19. Stroustrup B.: The Design and Evolution of C++ Addison-Wesley, 2004 (in Russian translation: Piter, St.Petersburg, 2007)
20. Carter J.B., Khandekar D.,Kamb L. 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
21. Chiba S.: A Metaobject Protocol for C++ Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pp 285–299, October 1995
22. Harris T., Fraser K. Language Support for Lightweight Transactions Object-Oriented Programming, Systems, Languages, and Applications. October 2003, 388402.
23. Herlihy M., Moss J.E.B. Transactional Memory: Architectural Support for Lock-Free Data Structures Proceedings of the 20th Annual International Symposium on Computer Architecture (1992) 289–300
24. Arslambekov R.M., Potemkin V.A., Guccione S. Parallel version of MultiGen for multi-conformational analysis of biological activity of compounds XII International Conference CMMASS'2003, Book of abstracts ;
25. Kornev A. On globally stable dynamic processes Russian Journal of Numerical Analysis and Mathematical Modelling, **17**, No. 5, p 472