T-system — a high-performance automatic parallelization tool

A. Vodomerov

14th May 2005

Distributed computing

Our goal is to create high-performance large-scale distributed applications.

GRID gives the appropriate middleware.

The Globus toolkit connects many different computational resources and make it available through the single interface.

But how can we write applications for this?

At the moment, only simple statical MPI-oriented applications is supported. User writes RSL and send it to the job manager that schedules application.

Static parallelization

The tradional approach for parallel programming is a static parallelization based on MPI (Message-Passing-Interface).

The main properties:

- - the course of computation;
  - time needed for different parts of the program;
  - the location of all data in the program

  are all assumed to be known before computation.
- All nodes have the same hardware, performance.
- The set of working nodes is set at start and cannot be changed later.

The disadvantages of static parallelization

Use of MPI for modern challenges raises some problems:

- The order of calculations is not known apriori in programs with complicated logic (e.g. games, modelling)
- Working nodes can broke up, new nodes can be added
- CPUs with different speed cause inefficient work distribution
- Very low performance in loosely-coupled configurations (e.g. GRID)

Dynamic parallelization

Dynamic parallelization distribute computational work beetween nodes in run-time.

The system has a feedback: if some node becomes overloaded or underloaded, the tasks can be easily rebalanced.

This approach allows to avoid idle times, results in very high utilization. It solves the most problems of static parallelization.

T-functions

T-functions are functions that can be delayed and migrated between nodes.

An example:

```
x = f(a);
y = g(b);
z = x + y;
```

The value of `x` is not needed in the process of calculating `g(b)`. So `f` and `g` can be computed at the same time. Having that, `f` can be "moved" to different node and computed in parallel.

How programs are written with T-system:

1. the program is divided into parts that can work in parallel (the ones without data dependencies);
2. these parts of code are separated as T-functions.

Not-ready values

Let's have a closer look at the example:

```
x = f(a);  // line 1
y = g(b);  // line 2
```

The C/C++ programs are executed in **line-by-line** manner. First, line 1 is performed, than, line 2 is performed.

Question: What value does variable `x` hold just before 2nd line?

- On the one hand, there is no value — it is not computed yet.
- On the other hand, there is some value — it is in some way related with function `f`, and after some time the value will appear.

This is a so-called not-ready value.

T-variables

Usual C/C++ variables cannot hold such values. Special T-variables are used instead.

T-variable of type A (where A can be int, double, ...) is a variable, that can hold either a value of type A, or a not-ready value (but not both).

In current openTS version not-ready values are implemented as references to another T-variables.

TC syntax

C/C++ language has been extended with special modifiers (tval, tfun etc) for writing parallel programs. The result language got name TC.

The simplest program on TC:

```
tfun int fib(int n)
{
  if (n < 2) return n;
  tval int x = fib(n-1);
  tval int y = fib(n-2);
  return x + y;
}
```


Correctness

First of all, any automatic parallelization system should work correctly.

A parallelized program must yield the same result as the original one.

TC language is as close to C++ as possible. After adding tval, tfun modifiers to working C++ program it's result doesn't change.

It is not true in all programs, but all violations have very good reasons. E.g. precise implementation would be much less efficient.

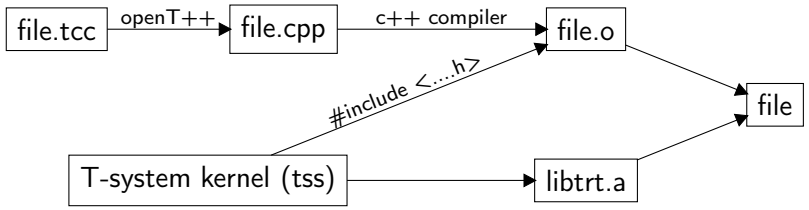
New keywords implementation

The support for new keywords is implemented in the following way:

- for each modifier, the corresponding template class is defined, e.g TVar<type> for tval type and TFun<type> for T-functions whose return value type is type.
- special preprocessor openT++ parses full C/C++ syntax and expand modifiers into their template counterparts;
- the implementation of this template classes, as well as some supporting mechanisms (scheduler, transport layer etc) are defined in T-system kernel

T-program compilation

How T-programs get compiled?



T-functions implementation

Goals that must be met:

- transparent access by name to all function arguments;
- ability to control T-function activation;
- migration between nodes.

T-functions is implemented as C++ object, that inherits abstract base class `TFun` and overrides pure virtual method `work()`, which, in fact, does all the computations. It is also inherited from a structure, containing all arguments.

From the therotical point of view, T-functions is a closure.

T-functions implementation (example)

Consider example:

```
tfun double func(int x)
{
  return sqrt(x);
}
```

After openT++ preprocessing:

```
struct func_TFunArgs {
  int x;
};
class func_TFunImpl : public TFun<double>,
  public func_TFunArgs {
  virtual TVar<double> work();
};
TVar<double> func_TFunImpl::work() {
  return sqrt(x);
}
```


T-function call

When program call T-function it really execute a “stub”, that create new T-function and put it into task queue. The reference to function return value (which is not ready yet) is returned.

```
TVar<double> func(int x) {
  func_TFunImpl *f = new func_TFunImpl(x);
  taskqueue.put(f);
  return f->retval;
}
```

Access to non-ready data results in “sleep” (current task is put into the queue tail, and a next one is popped out).

T-variables implementation

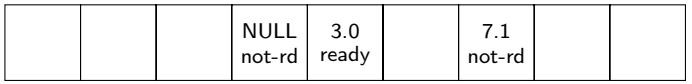
Main goal:
Provide transparent access to data from all nodes. T-variable can be created on one node and used from many others.

- Different address spaces — can not exchange pointers.
- The only available communication is a message passing.

The solution:
All data are stored in global array-like distributed storage and T-variables are just references (pointers) to its items.

Supermemory

Supermemory is a large array of cells. Informally, each cell contains one variable (value + state: ready-notready).



T-variable is just a cell index in the supermemory cell array. Given cell number, its data can be retrieved from all nodes.

Each cells reside on one node (master node). It controls cell: writes cell data, destroys it when it's not needed anymore.

Scheduler

The heart of any dynamic parallelization system is a scheduler. It determines the overall system performance.

The main scheduler's goal is to minimize program run time, avoid downtime.

The more precisely scheduler models program execution, the more efficient resource usage will be.

Scheduler is implemented as T-system extension. User can easily write his own scheduler and link his program with it.

Scheduler algorithm

Parameters affecting schedule decision:

- task complexity (flops);
- processor speed (flops);
- required data size (Mb);
- network bandwidth, latency (Mb/s, s).

Task time approximation:

$$t = \frac{flops}{cpu} + \frac{size}{bandwidth} + 2 \times latency$$

openTS scheduler minizes time, then equalizes node load:

$$T_{\max} \to \min, \qquad DT = \frac{1}{N}\sum T_i^2 - \left(\frac{1}{N}\sum T_i\right)^2 \to \min$$

GRID

GRID will be the basis for distributed computing in future.

Key points:

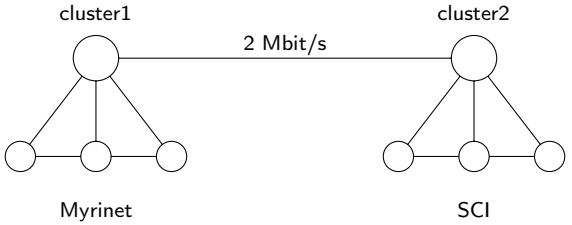
- Heterogeneity: different CPU, architectures, OS.
- Highly different communication channles. As a rule, the bandwidth of cluster interconnect is 100-1000 times bigger than links between clusters.
- Dynamically changed configuration: nodes are constantly adding and removing.

Static parallelization doesn't work anymore.

Automatic parallelization is the key to high-performance GRID computing.

T-system in GRID

How can T-system work in GRID?



Answer: it's enough to use metacluster MPI (PACX-MPI, MPICH-G2) as a transport.

T-system over MetaMPI

The naive answer is absolutely wrong!

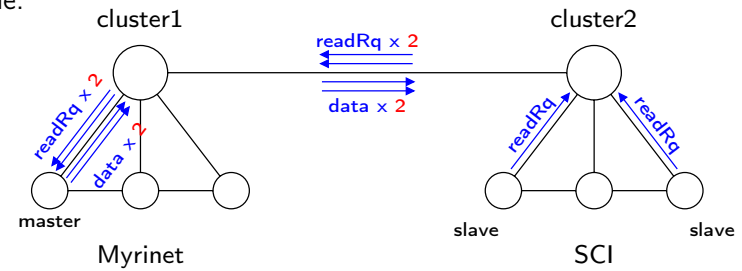
The reasons:

- MPI doesn't allow to remove or add nodes
- Enormous traffic between clusters
- Unscalable by number of nodes

The main problem: each cluster node communicates directly with all nodes of another cluster (supermemory, DRC, scheduler).

Message passing in metacluster

Suppose, a node from the second cluster need data from the first one.

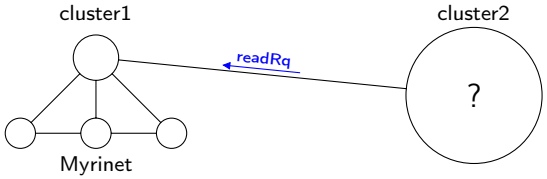


The size of transferring data is directly proportional to number of nodes.

$$W \sim N_2$$

Metacluster hierarchy

One should take into account real physical topology of metacluster in order to achieve efficiency.



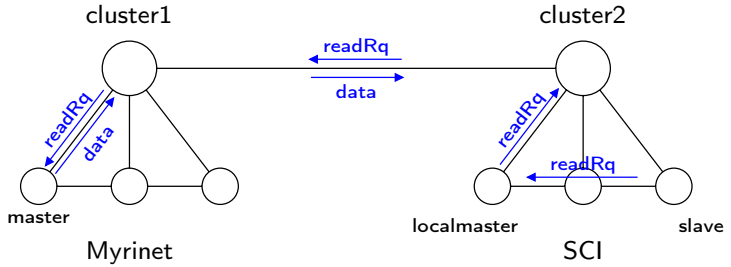
This approach allows to achieve greater scalability.

$$W = O(1)$$

Supermemory in GRID

Local cache is created to reduce the traffic between clusters.

Mixed local-master cell is set up for each remote supermemory cell.



Metacluster scheduling

Precise information about remote cluster load is not available (size and transfer time $\sim N_2$).

Two-stage scheduling:

1. each node schedule on local nodes and remote cluster as a whole;
2. if task is scheduled for remote cluster, the real node for it will be choosed on remote cluster.

Logical independence of clusters is achieved.

Data transfer in heterogenouse environment

```
tfun int func(int x, double y) { ... }
```

After preprocessor:

```
struct func_TFunArgs {
  int x;
  double y;
};
class func_TFunImpl : public TFun<double>,
  public func_TFunArgs {
  virtual TVar<double> work();
```

Such structures cannot be directly sent between different GRID nodes:

- different byte order (little-, big-endian);
- different base types sizes: `int`, `long`, `double`;
- different offset, alignment;
- different compiler generate different memory layout;
- pointer, classes with virtual methods.

Serialization

The network transfers raw bytes. C/C++ object must be converted to and from raw bytes.

This is known as (de)serialization.

All applications without any substantial changes now work on x86 + AMD64, Linux (x86) + Win32 (x86)!

GRID integration

Now, we have T-system efficiently running in the GRID environment. How we can achieve flexibility?

Two additional services are added to base GRID services:

- T-application monitoring service
- T-application resource management service

T-application monitoring

The first one collects information from running and queued T-system application:

- name, version, path to executable
- username, priority, time of start, estimated finish time
- T-function, memory and message statistics
- application sensivity to network latency, bandwith

Resource management for T-application

The second service use the information from the first one. It distribute available resource between application. When new resource becomes available, application to receive it is selected according to the priority and sensitivity.

It is very important that we can give resource to and take it back from application in run-time. The task can migrate over network, new nodes can be immediately added to the application that benefit from it most. When a new high-priority application arrives, we can drawback resources from the running one.

We achieve a level of resource management flexibility and efficiency that matches GRID distributed nature.

The end

Thank you for your attention!

Alexander Vodomerov, MSU IMEC
alex@sectorb.msk.ru