СУПЕРКОМПЬЮТЕРНЫЙ КОНСОРЦИУМ УНИВЕРСИТЕТОВ РОССИИ

МОЛОДЕЖНАЯ ШКОЛА
«Суперкомпьютерные технологии в науке, промышленности и образовании»

«Есть ли жизнь после MPI?»

Абрамов С.М.

д.ф.-м.н., чл.-корр. РАН,

ИПС имени А.К.Айламазяна РАН

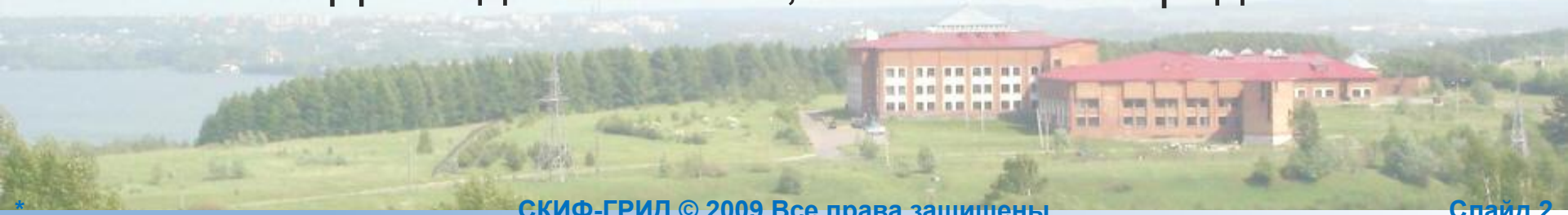
ННГУ им. Н.И. Лобачевского, Нижний Новгород, 26-31 октября 2009 г.

Есть ли жизнь после MPI?

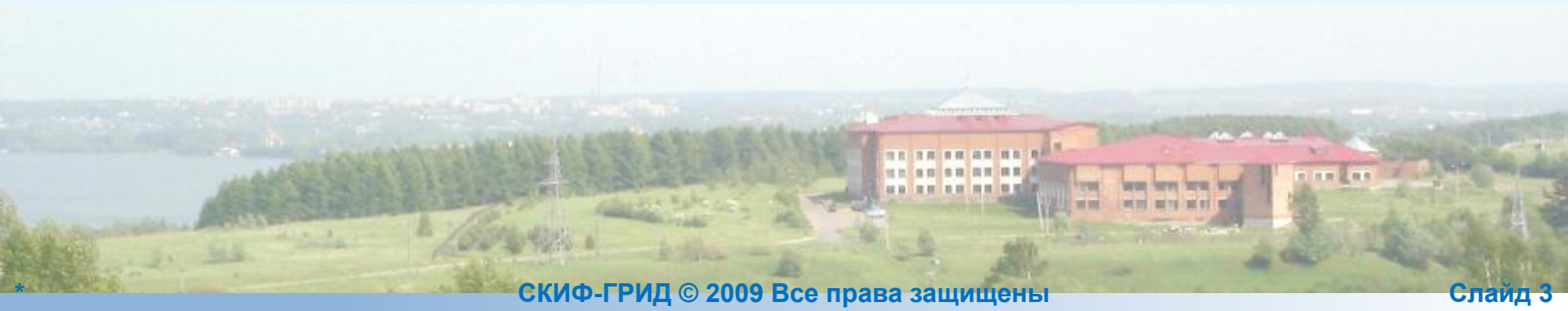
Абрамов С.М.†, Климов А.В.‡,
Лацис А.О.‡, Московский А.А.†

†ИПС имени А.К.Айламазяна РАН
‡ИПМ имени М.В.Келдыша РАН

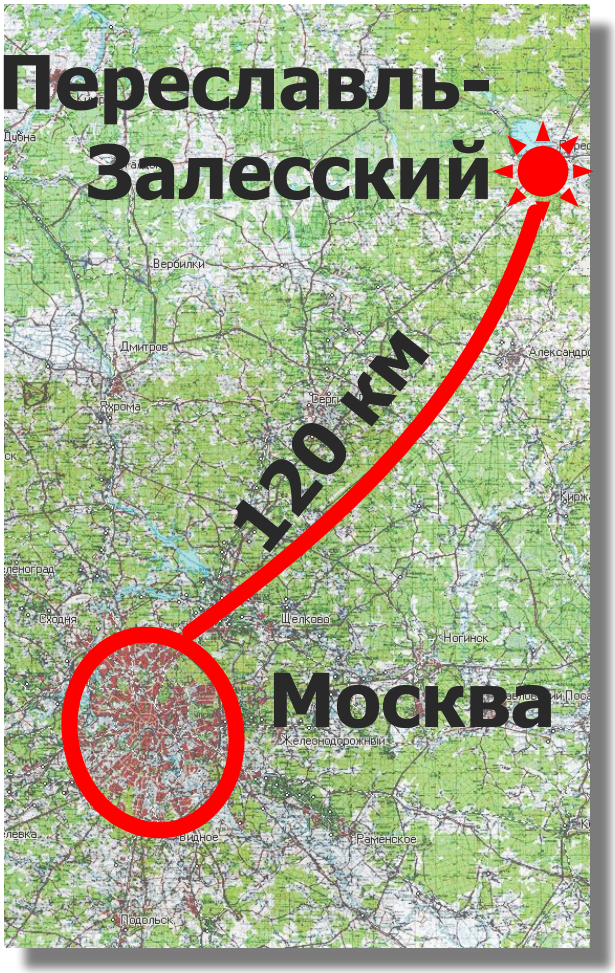
Доклад 2009.10.31, Нижний Новгород



Переславль-Залесский. Институт программных систем имени А.К.Айламазяна Российской академии наук



Переславль-Залесский



- Красивый старинный (860 лет) город России на берегу Плещеева озера
- Центр Золотого кольца
- Родина Св.Александра Невского, родина многих великих князей
- Здесь Петр Великий создавал свою первую «потешную флотилию» — место рождения Российского флота
- Древний центр Российской Православной Церкви

ИПС имени А.К.Айламазяна РАН, Переславль-Залесский

Основание Института

- Основан в 1984 году по постановлению ВПК для развития информатики и вычислительной техники в стране
- Первый директор (1984–2003) — проф. А.К.Айламазян
- В декабре 2008 Институту присвоено имя А.К.Айламазяна

2009: Организационная структура института

- Исследовательский центр искусственного интеллекта
- Исследовательский центр медицинской информатики
- Исследовательский центр мультипроцессорных систем
- Исследовательский центр системного анализа
- Исследовательский центр процессов управления
- Научно-образовательный центр — Международный детский компьютерный лагерь (МДКЦ) имени А.К.Айламазяна

Университет города Переславля имени А.К.Айламазяна

Программы «СКИФ» и «СКИФ-ГРИД»

- Заказчики-координаторы
  - НАН Беларуси
  - Агентство «Роснаука»
- Головные исполнители
  - Объединенный институт проблем информатики НАН Беларуси
  - Институт программных систем РАН
- Исполнители
  - «СКИФ» 2000-2004 — 10+10 организаций Беларуси и России
  - «СКИФ-ГРИД» 2007-2010 — 10+20 организаций Беларуси и России
- 2003-2008: 5 суперЭВМ семейства «СКИФ» в рейтинге Top500

Семейство суперЭВМ «СКИФ»: Ряды 1, 2, 3 и 4

Сделано: Ряды 1–3

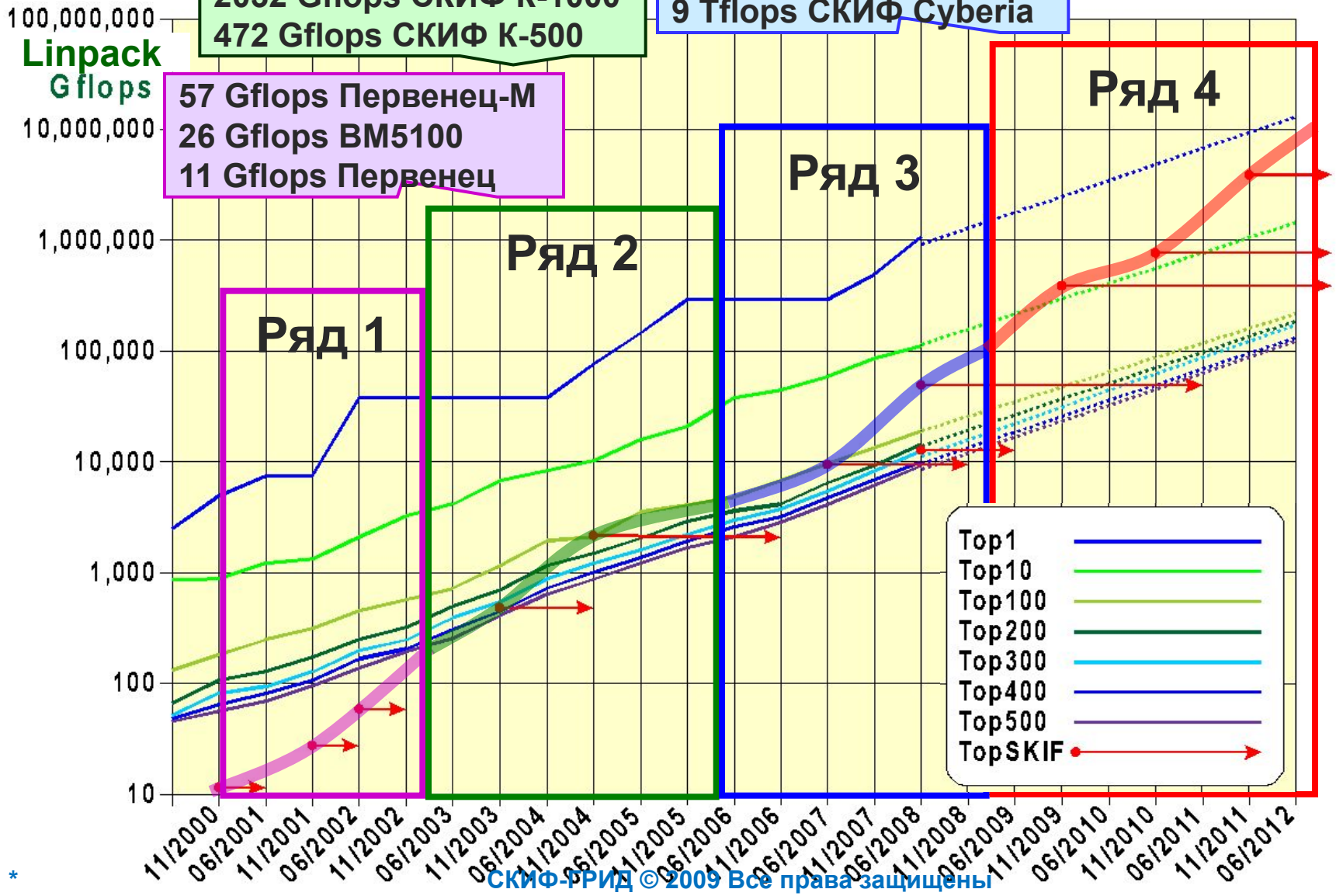
Ближайшие планы: Ряд 4

1 кв.2012 СКИФ П~5.0
3 кв. 2010 СКИФ П-1.0
3 кв. 2009 СКИФ П-0.5

47.17 Tflops СКИФ МГУ
12.2 Tflops СКИФ Урал
9 Tflops СКИФ Cyberia

2032 Gflops СКИФ К-1000
472 Gflops СКИФ К-500

57 Gflops Первенец-М
26 Gflops ВМ5100
11 Gflops Первенец

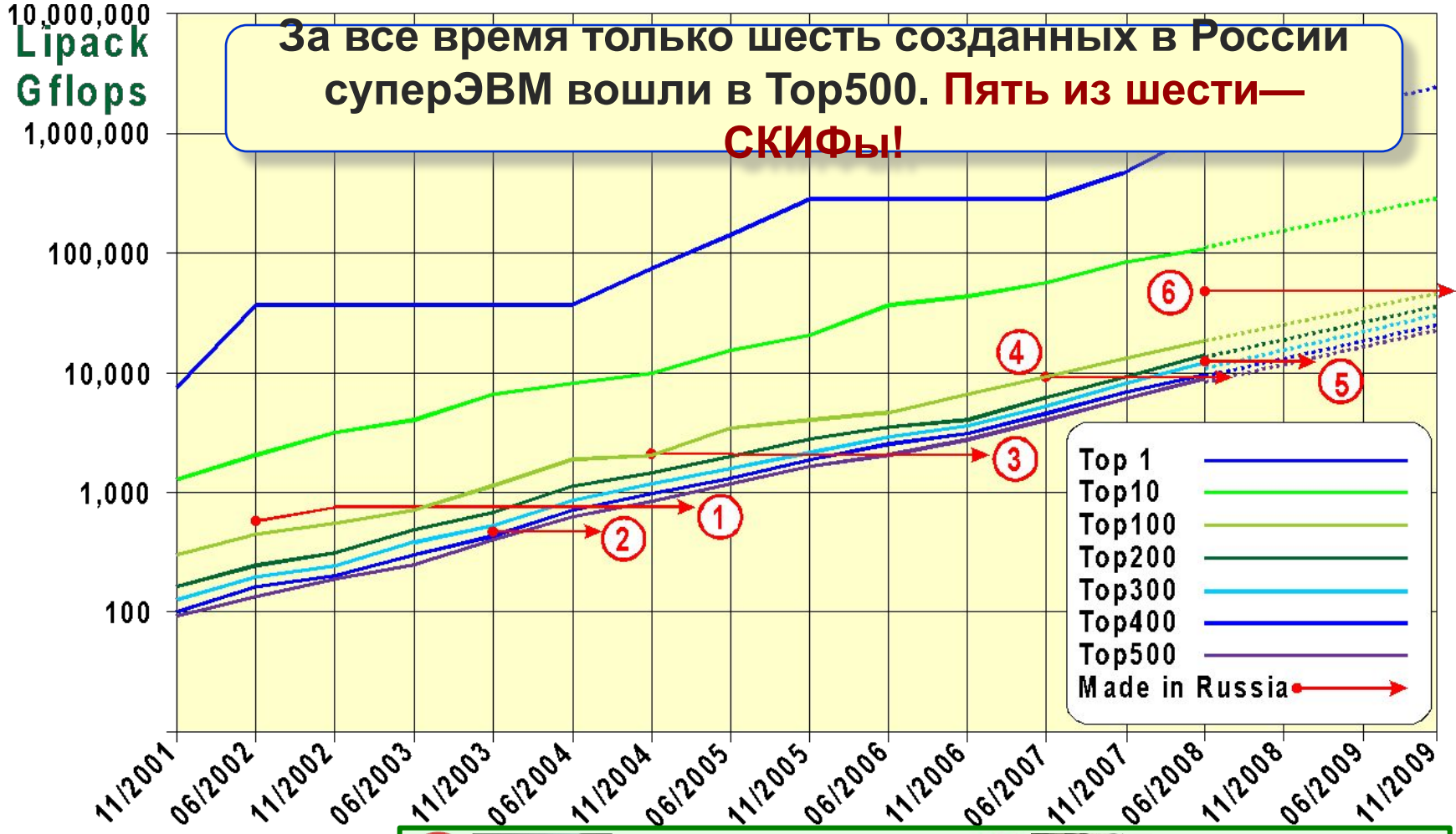


*

ИПС РАН

СКИФ ГРИД

За все время только шесть созданных в России суперЭВМ вошли в Top500. Пять из шести— СКИФы!



1 2002 июнь
МВС 1000М
0.734/1.024 TFlops

3 2004 ноябрь
СКИФ К-1000
2.032/2.534 TFlops

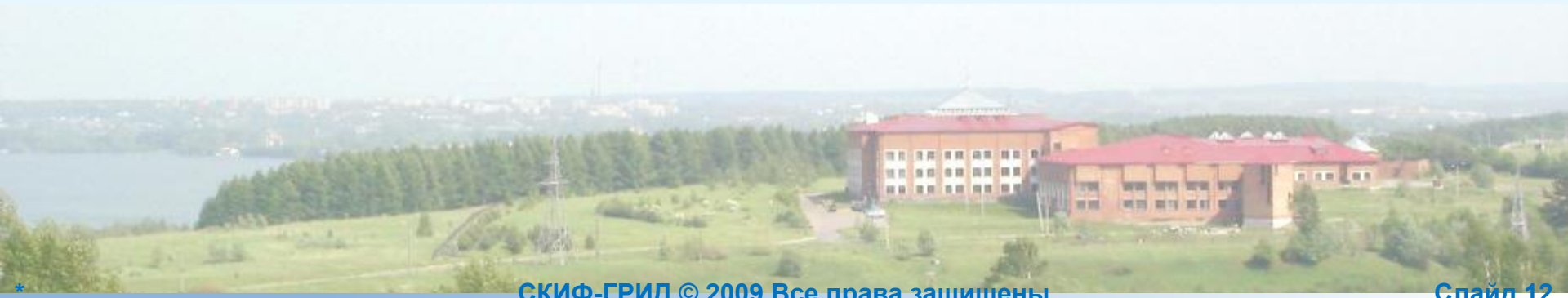
5 2008 май
СКИФ Урал
12.2/15.9 TFlops

2 2003 ноябрь
СКИФ К-500
0.423/0.717 TFlops

4 2007 февраль
СКИФ Cyberia
9.013/12.002 TFlops

6 2008 май
СКИФ МГУ
47.1/60 TFlops

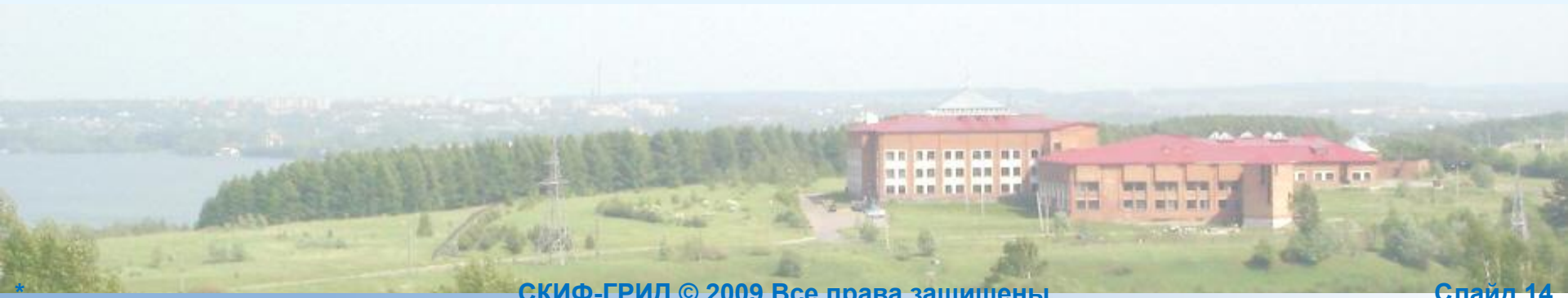
Что затрудняет эффективное использование MPI в суперЭВМ ближайшего будущего?



Проблемы MPI

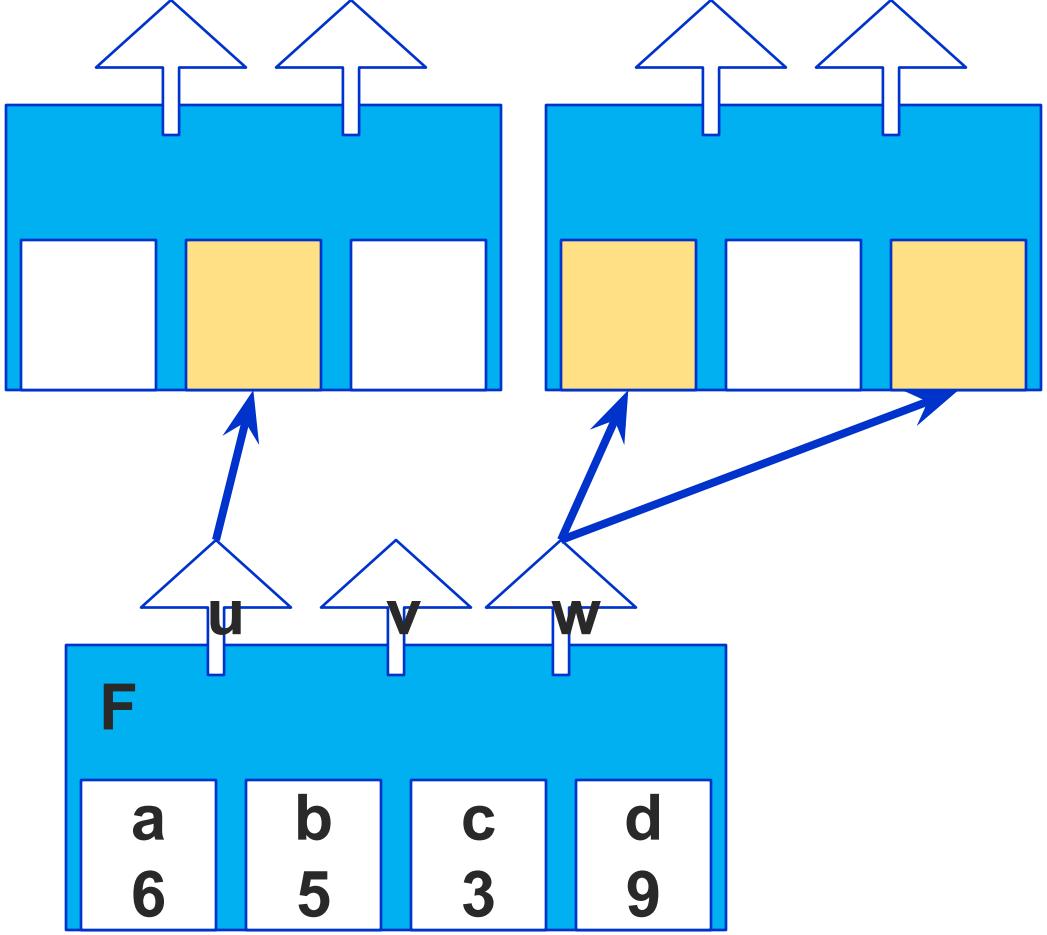
- Рост числа процессоров (и ядер) в суперЭВМ будет продолжаться
  - Сегодня 1 Pflops ≈ 20,000 CPU ≈ 80,000 ядер
  - Установки с 1,000,000 ядрами появятся очень скоро
- Трудности эффективной реализации MPI для гигантского числа вычислительных узлов
  - да еще и многоядерных!
- Трудности эффективного использования программистами MPI для случая гигантского числа вычислительных узлов
- Есть разрыв между тем, что реализует аппаратура (SMP + односторонние обмены) и тем, что имеется в MPI (двусторонние обмены, взаимодействие по каналу)

Т-система: автоматическое динамическое распараллеливание программ

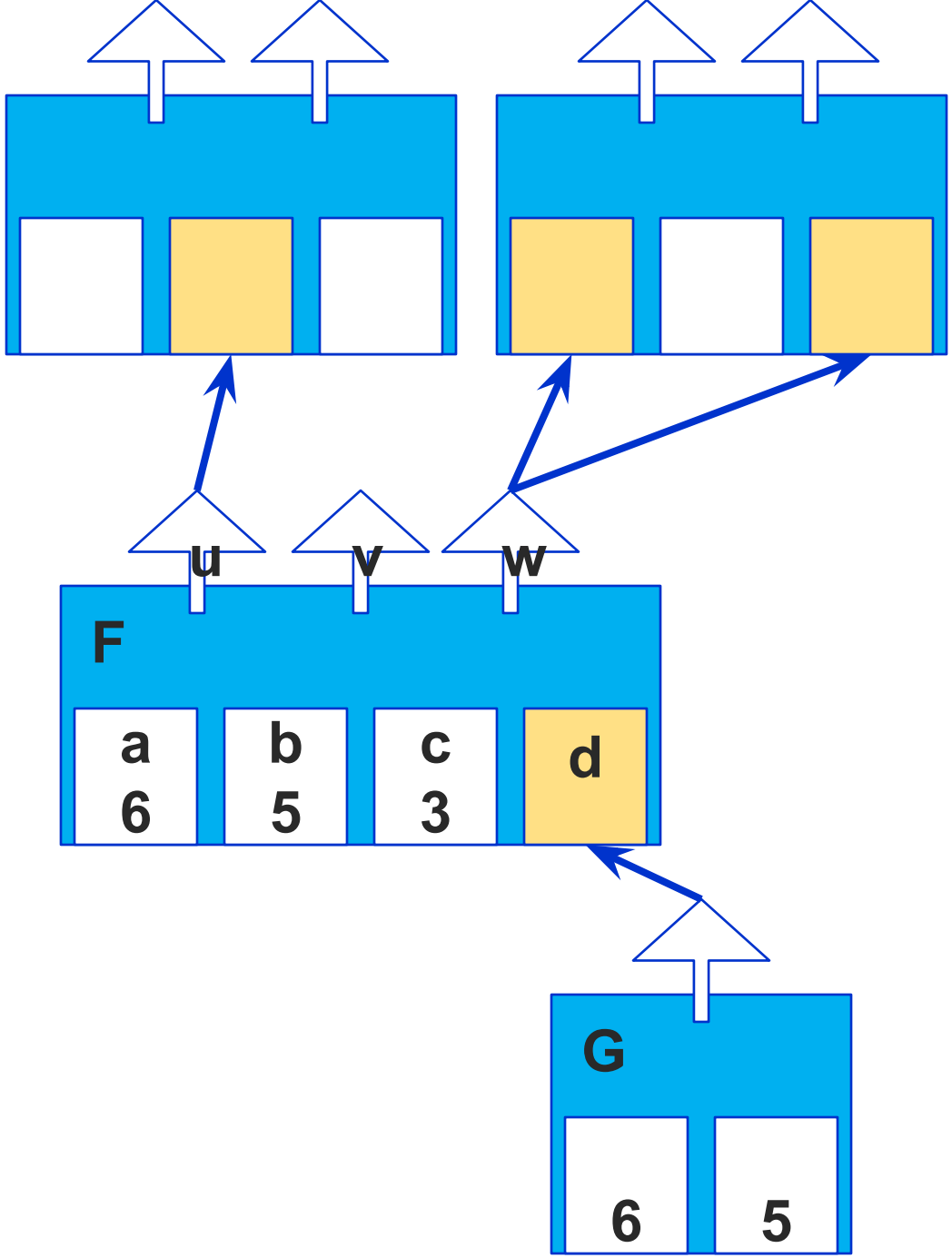


Т-система (неформально)

- Функциональная модель + императивное описание тела функции
- Арность и коарность функций
- Готовые и неготовые значения
- Вызов Т-функции — порождение процесса
- Можно копировать неготовые значения, в том числе и передавать их как результат
- Любые иные операции с неготовым значением приводит к «засыпанию» процесса на данной Т-переменной
- Побудка будет, когда Т-переменная примет готовое значение
- Состояние вычисления: сеть из процессов (ребра — отношение «поставщик-потребитель»), процесс исполнения: автоматформальной сети




```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```



...

d = G(a, b)

b = d

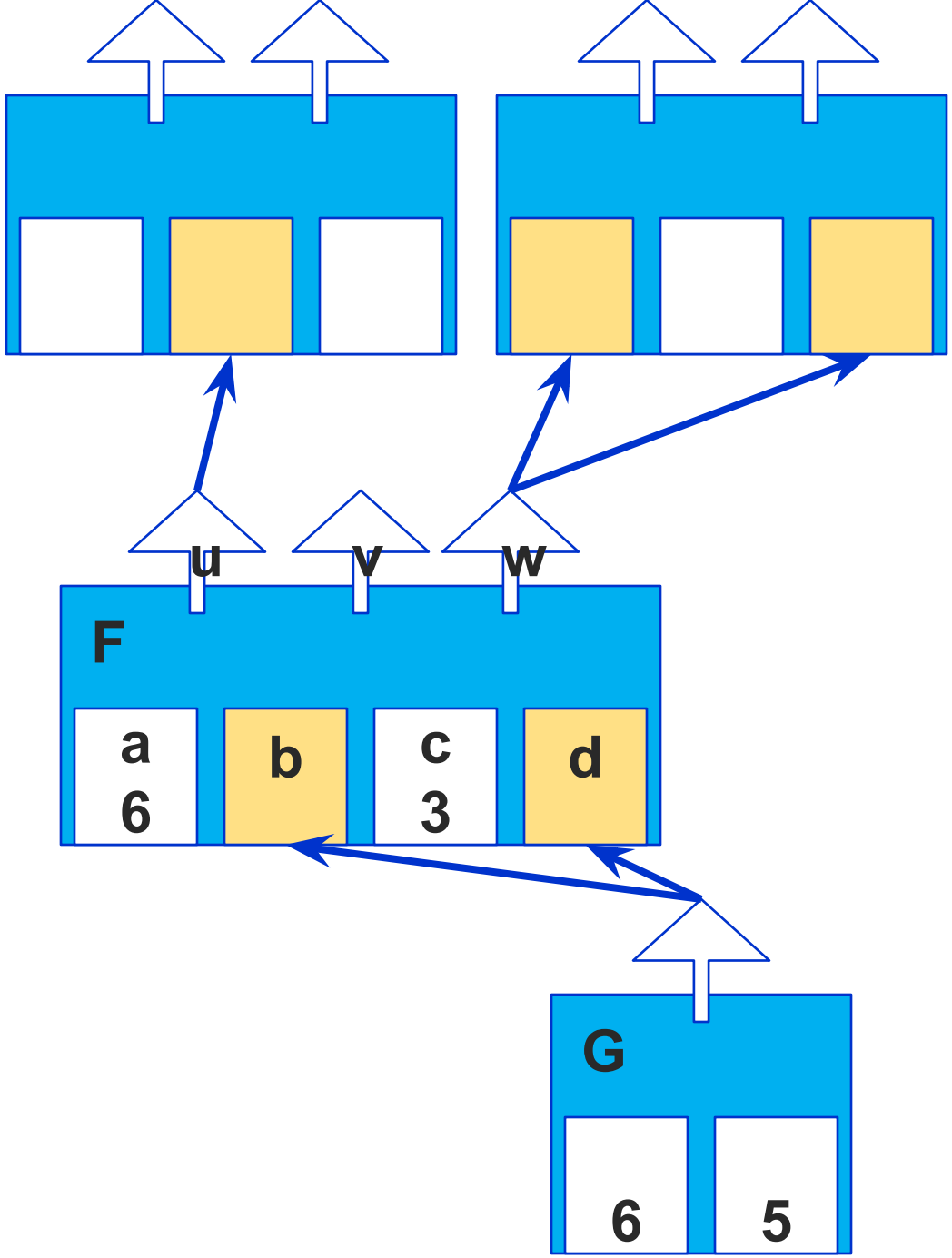
u = a

d = c

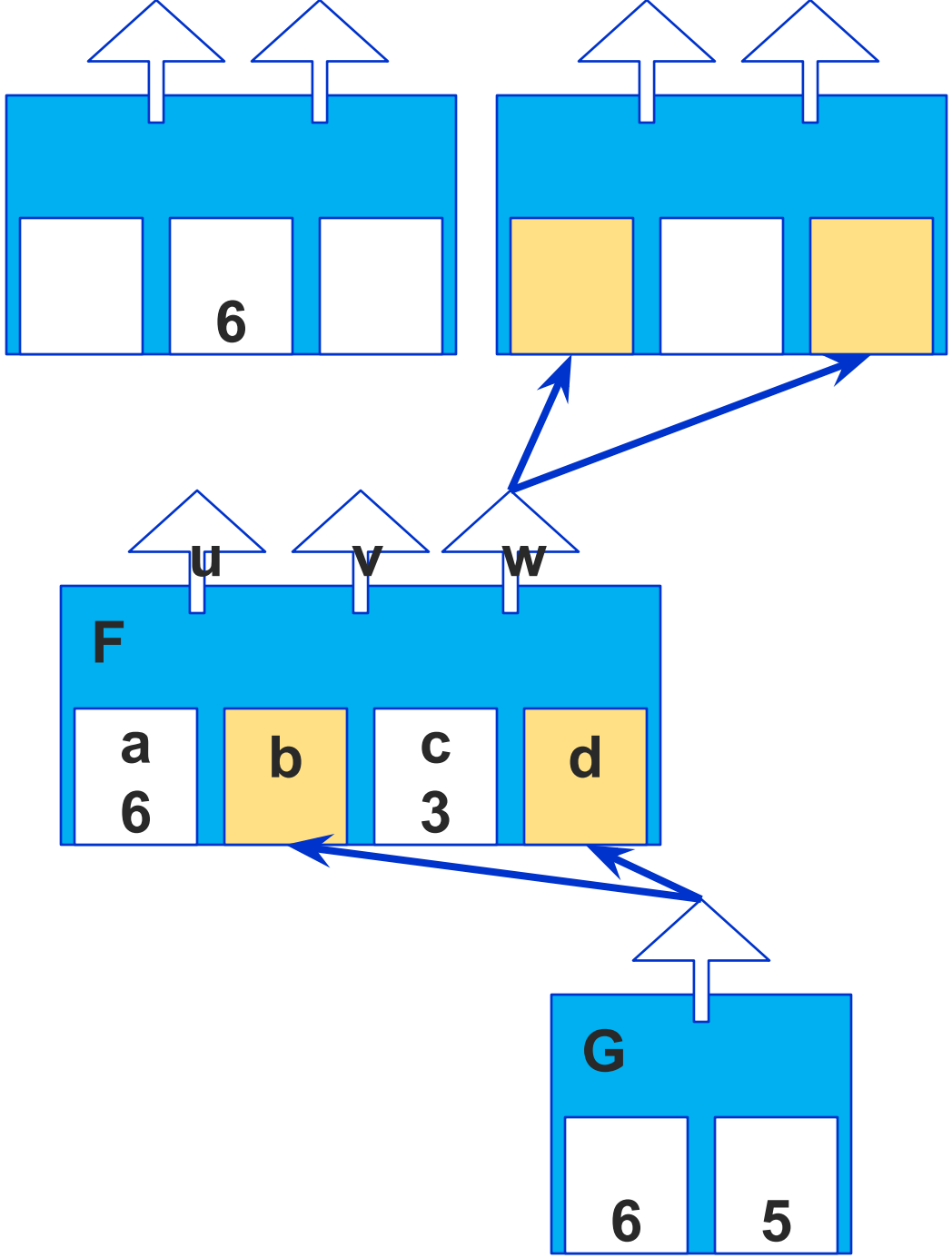
v = a

w = b

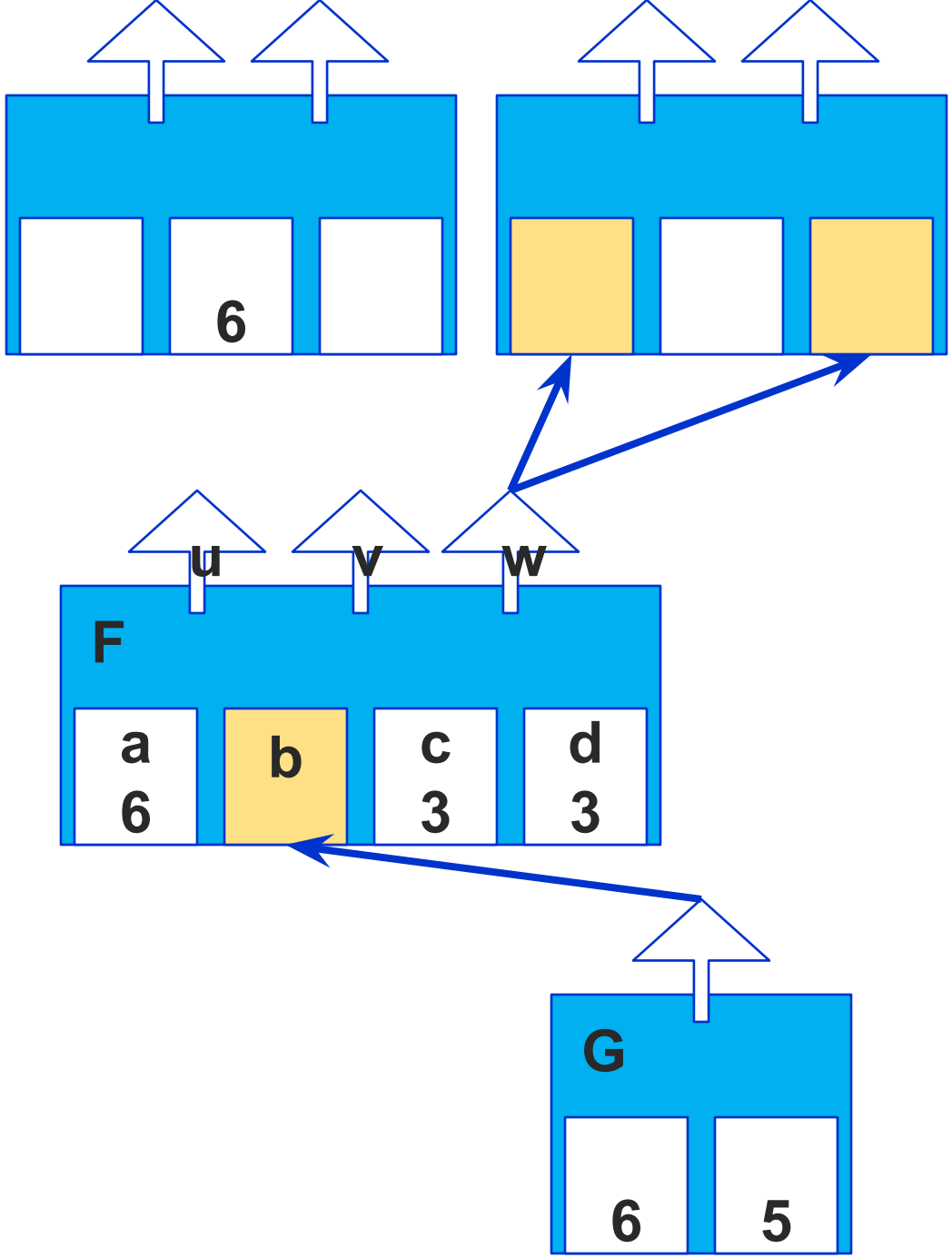
...



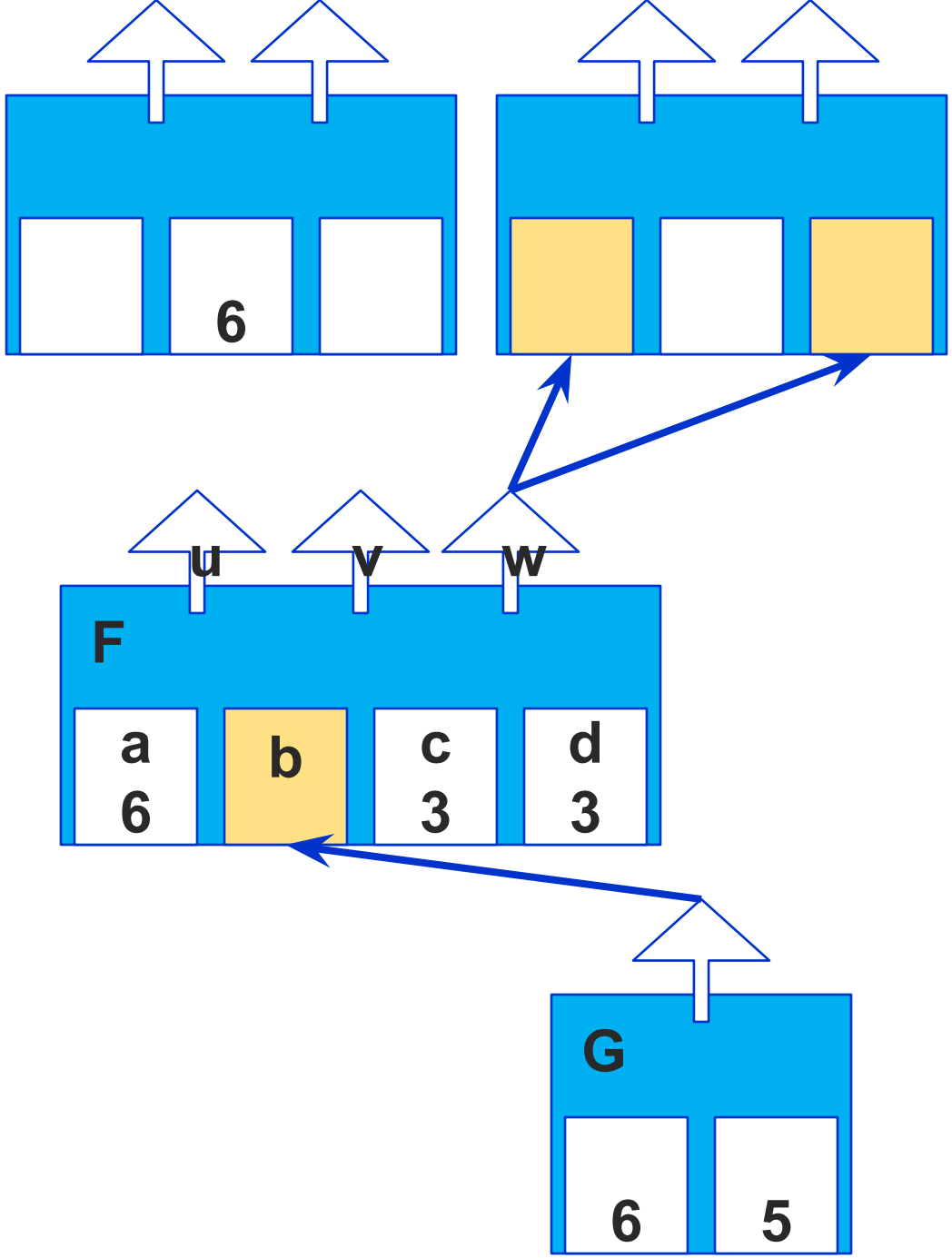

```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```



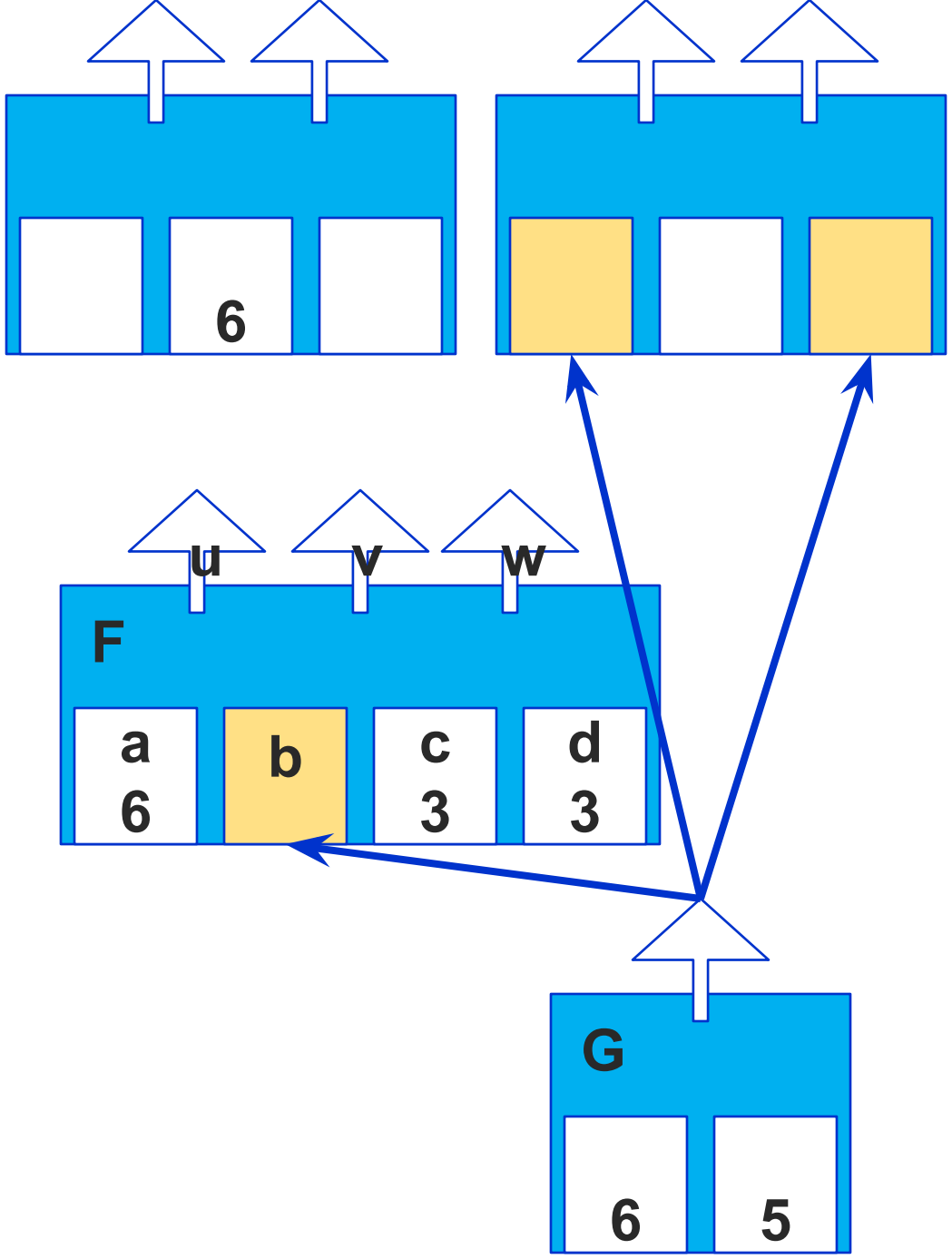

```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```



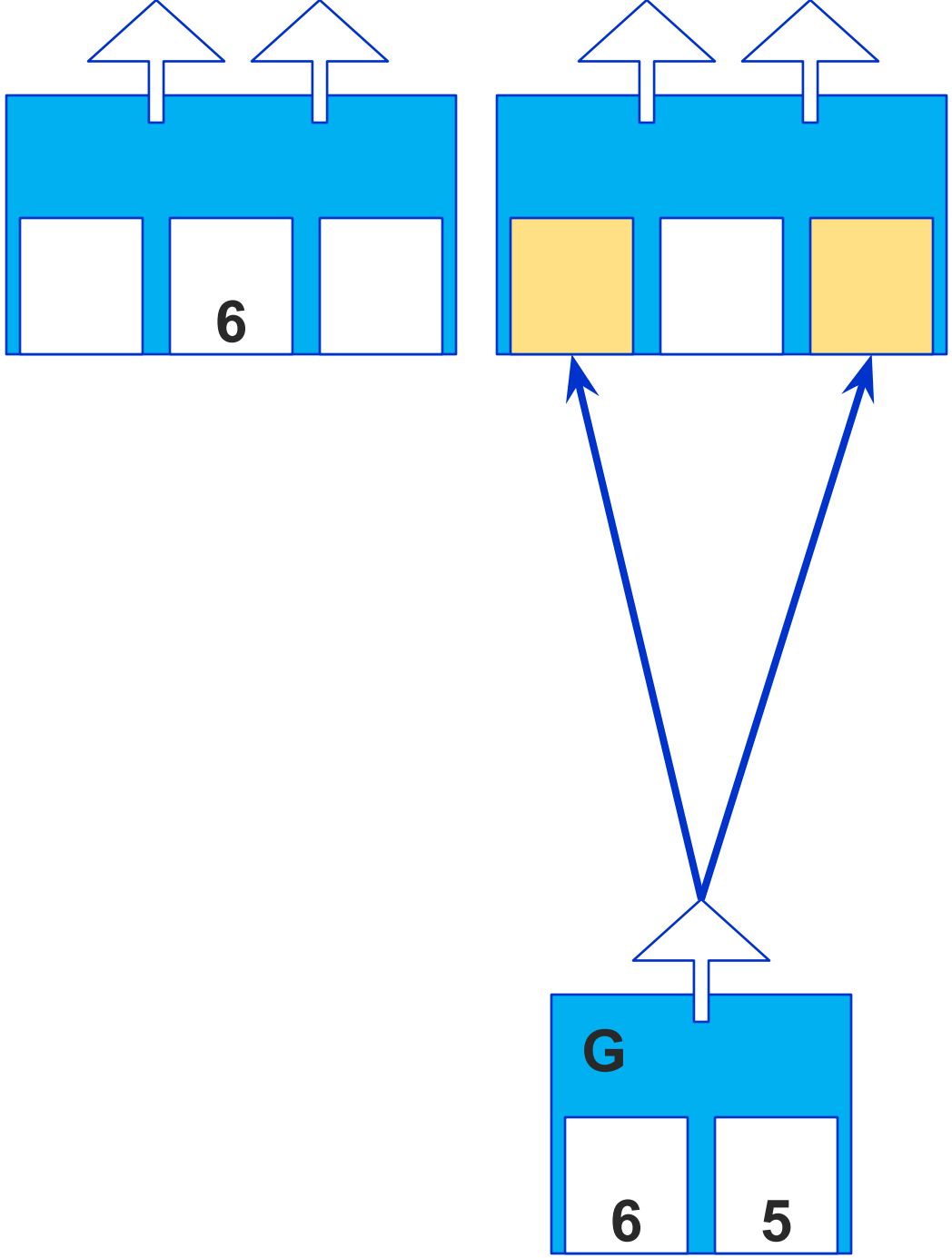

```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```




```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```




```
...
d = G(a, b)
b = d
u = a
d = c
v = a
w = b
...
```



T-System History

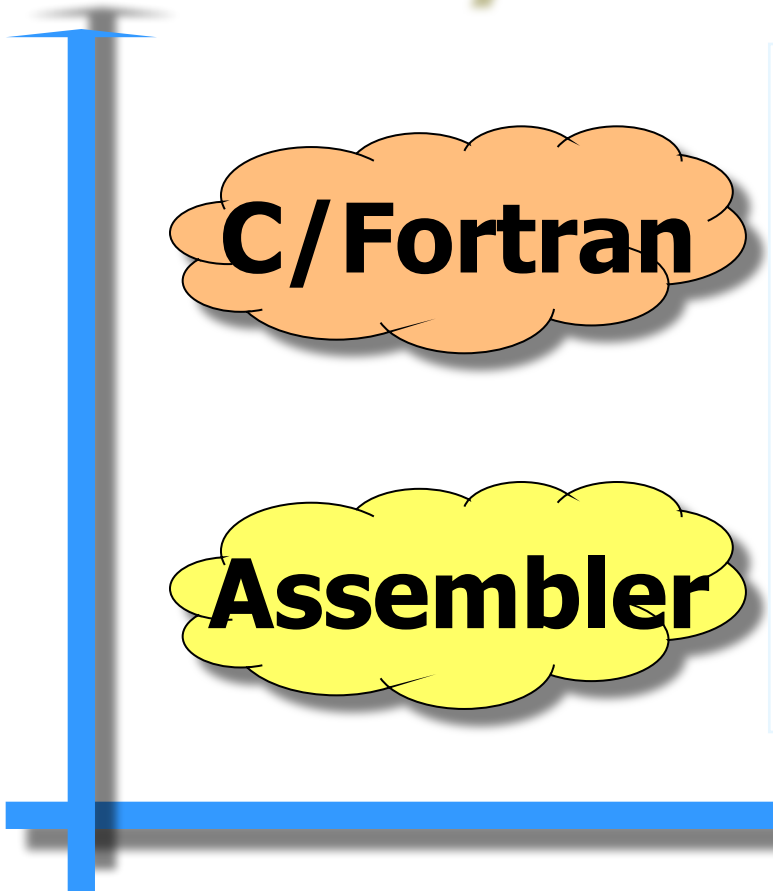
- **Mid-80-ies**
  Basic ideas of T-System
- **1990-ies**
  First implementation of T-System
- **2001-2002, "SKIF"**
  GRACE — Graph Reduction Applied to Cluster Environment
- **2003-current, "SKIF"**
  Cooperation with Microsoft
  Open TS — Open T-system

Open TS Overview

Comparison: T-System and MPI

High-level
a few keywords

Low-level
hundred(s) primitives



Sequential

T-System in Comparison

Related work	Open TS differentiator
Charm++	FP-based approach
UPC, mpC++	Implicit parallelism
Glasgow Parallel Haskell	Allows C/C++ based low-level optimization
OMPC++	Provides both language and C++ templates library
Cilk	Supports SMP, MPI, PVM, and GRID platforms

Open TS: an Outline

- High-performance computing
- “Automatic dynamic parallelization”
- Combining functional and imperative approaches, high-level parallel programming
- T++ language: “Parallel dialect” of C++ — an approach popular in 90-ies

T-Approach

- "Pure" functions (**tfunctions**) invocations produce grains of parallelism
- T-Program is
  - Functional – on higher level
  - Imperative – on low level (optimization)
- C-compatible execution model
- Non-ready variables, Multiple assignment
- "Seamless" C-extension (or Fortran-extension)

T++ Keywords

- **tfun** — T-function
- **tval** — T-variable
- **tptr** — T-pointer
- **tout** — Output parameter (like &)
- **tdrop** — Make ready
- **twait** — Wait for readiness
- **tct** — T-context

Short Introduction (Sample Programs)

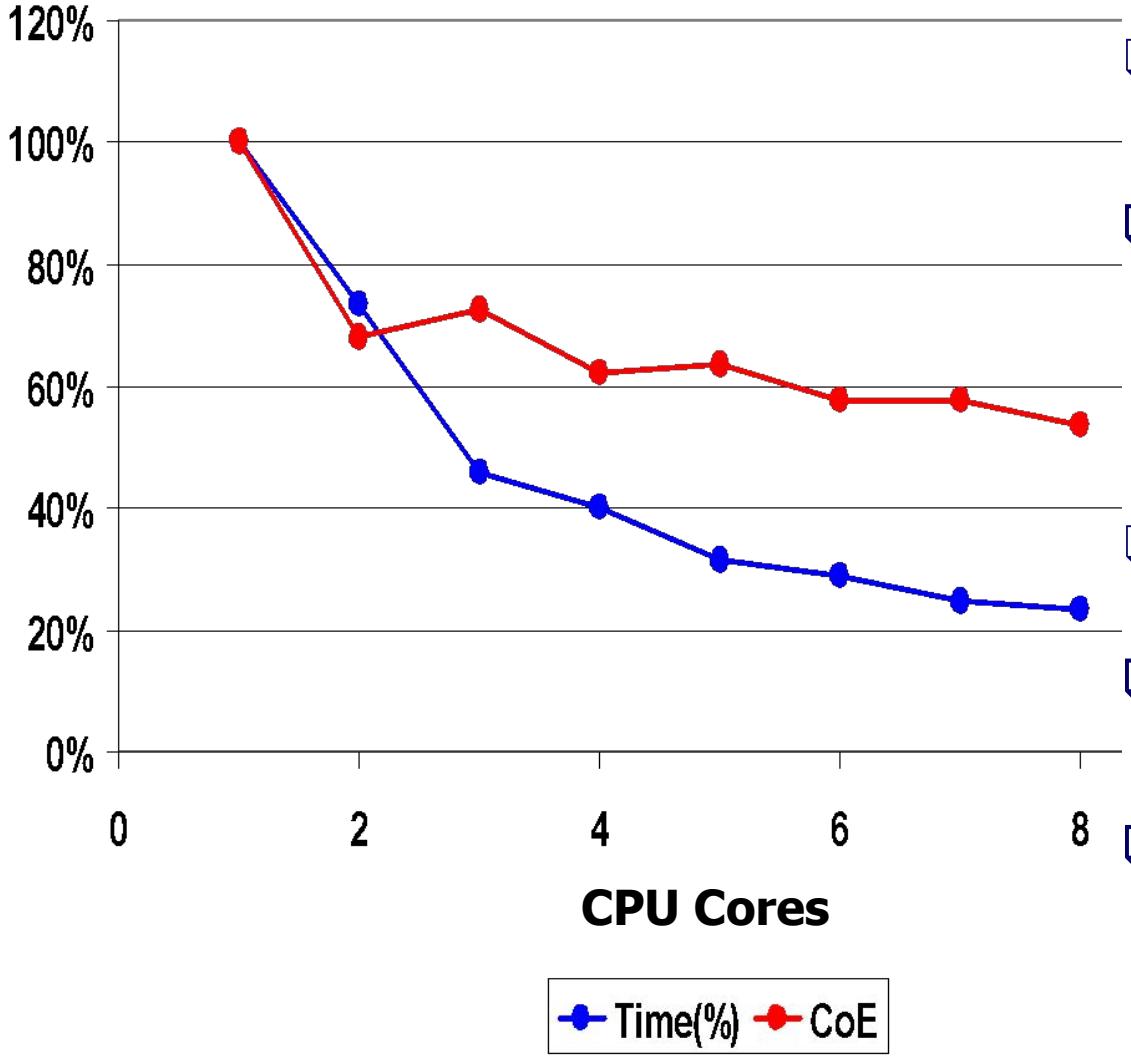
Sample Program (C++)

```
#include <stdio.h>

int fib (int n) {
    return n < 2  ?  n : fib(n-1)+ fib(n-2);
}

int main (int argc, char **argv) {
    if (argc != 2) { printf("Usage: fib <n>\n"); return 1; }
    int n = atoi(argv[1]);
    printf("fib(%d) = %d\n", n, fib(n));
    return 0;
}
```

Sample Program (T++)

```
#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) { printf("Usage: fib <n>\n"); return 1; }
    int n = atoi(argv[1]);
    printf("fib(%d) = %d\n", n, (int)fib(n));
    return 0;
}
```

Sample Program (T++)



- WinCCS cluster, 4 nodes
- **CPU:** AMD Athlon 64 X2 Dual Core Processor 4400+ 2.21 GHz
- Gigabit Ethernet
- **time% =** $time_{tapp}(N)/timet_{app}(1)$
- **CoE =** 1/(n×time%)

Approximate calculation of Pi (C++)

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
double
    isum(double begin, double
    finish, double d) {
  double dl = finish - begin;
  double mid =
    (begin + finish) / 2;

  if (fabs(dl) > d)
    return isum(begin, mid, d)
    + isum(mid, finish, d);
  return f(mid) * dl;
}
```

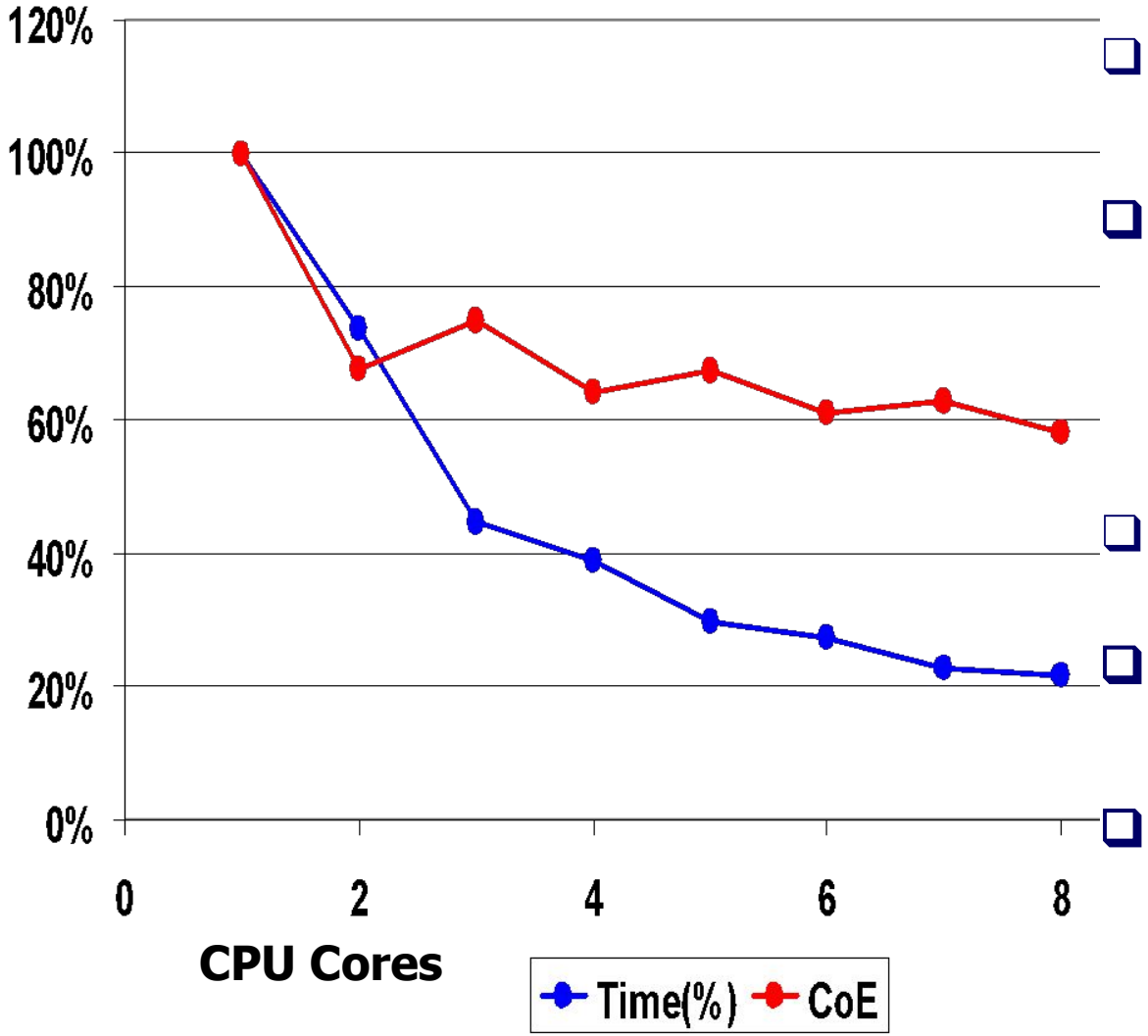
```
double f(double x) {
  return 4/(1+x*x);
}
int main(int argc, char* argv[]){
  unsigned long h;
  double a, b, d, sum;

  if (argc < 2) {return 0;}
  a = 0; b = 1; h = atol(argv[1]);
  d = fabs(b - a) / h;
  sum = isum(a, b, d);
  printf("PI is approximately
    %15.15lf\n", sum);
  return 0;
}
```

Approximate calculation of Pi (T++)

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
tfun double
     isum(double begin, double
     finish, double d) {
   double dl = finish - begin;
   double mid =
     (begin + finish) / 2;

   if (fabs(dl) > d)
     return isum(begin, mid, d)
     + isum(mid, finish, d);
   return (double)f(mid) * dl;
}
```

```
tfun double f(double x) {
   return 4/(1+x*x);
}
tfun int main(int argc, char* argv[]){
   unsigned long h;
   double a, b, d, sum;

   if (argc < 2) {return 0;}
   a = 0; b = 1; h = atol(argv[1]);
   d = fabs(b - a) / h;
   sum = isum(a, b, d);
   printf("PI is approximately
     %15.15lf\n", sum);
   return 0;
}
```

Calculation of Pi (T++)



- WinCCS cluster, 4 nodes
- **CPU:** AMD Athlon 64 X2 Dual Core Processor 4400+ 2.21 GHz
- Gigabit Ethernet
- **time% =** $time_{tapp}(N)/timet_{app}(1)$
- **CoE =** $1/(n \times time\%)$

Map-Reduce

----- Original Message -----
From: Alexy Maykov
Sent: Monday, October 02, 2006 11:58 PM
Subject: MCCS projects

…
I work in Microsoft Live Labs … I have several questions below:

1. How would you implement Map-Reduce in OpenTS?

…

Map-Reduce (C++)

```
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
#include <ctime>
using namespace std;

int fib (int n)
{
   return (n < 2) ? n : fib(n-1) + fib(n-2);
}
int plus (int val1, int val2)
{
   return val1 + val2;
}
int main (int argc, char *argv[ ])
{
   const int factor = 23;
   const int vector_size = 40;
   vector<int> a, b, c;
   vector<int> fa, fb;

   cout << " Filling vectors..." << endl;
   for (int i = 1; i <= vector_size; i++)
   {
      a.push_back(i % factor);
      b.push_back((vector_size + 1 - i) % factor);
      c.push_back(0);
      fa.push_back(0);
      fb.push_back(0);
   }

   cout << " Mapping..." << endl;
   transform(a.begin(),  a.end(),  fa.begin(), fib);
   cout << " Mapping..." << endl;
   transform(b.begin(),  b.end(),  fb.begin(), fib);
   cout << " Reducing..." << endl;
   transform(fa.begin(), fa.end(), fb.begin(), c.begin(),
       ::plus);

   cout << endl << " Result: (" ;
   ostream_iterator<int> output(cout, " ");
   copy(c.begin(), c.end(), output);
   cout << "\b)" << endl;

   return 0;
}
```

Map-Reduce (C++)

```
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
#include <ctime>
using namespace std;

int fib (int n)
{
   return (n < 2) ? n : fib(n-1) + fib(n-2);
}
int plus (int val1, int val2)
{
   return val1 + val2;
}
int main (int argc, char *argv[ ])
{
   const int factor = 23;
   const int vector_size = 40;
   vector<int> a, b, c;
   vector<int> fa, fb;
```

```
cout << " Filling vectors..." << endl;
for (int i = 1; i <= vector_size; i++)
{
   a.push_back(i % factor);
```

Fibonacci

```
cout << " Mapping..." << endl;
transform(a.begin(),  a.end(),  fa.begin(), fib);
cout << " Mapping..." << endl;
transform(b.begin(),  b.end(),  fb.begin(), fib);
cout << " Reducing..." << endl;
transform(fa.begin(), fa.end(), fb.begin(), c.begin(),
    ::plus);

cout << endl << " Result: (" ;
ostream_iterator<int> output(cout, " ");
```

Just "Plus"

Map-Reduce (C++)

```
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
#include <ctime>
using namespace std;

int fib (int n)
{
}
int
{
}
int main (int argc, char *argv[ ])
{
    const int factor = 23;
    const int vector_size = 40;
    vector<int> a, b, c;
    vector<int> fa, fb;
```

```
cout << " Filling vectors..." << endl;
for (int i = 1; i <= vector_size; i++)
{
    a.push_back(i % factor);
    b.push_back((vector_size + 1 - i) % factor);
    c.push_back(0);
    fa.push_back(0);
    fb.push_back(0);
}

cout << " Mapping..." << endl;
transform(a.begin(),  a.end(),  fa.begin(), fib);
...." << endl;
(b.begin(),  b.end(),  fb.begin(), fib);
...." << endl;
(), fa.end(), fb.begin(), c.begin(),

Result: (" ;

    return 0;
}
```

Filling vectors:

a=[k%23 | k□[1..40]]

b=[(41-k)%23 | k□[1..40]]

Five vectors: a, b, fa, fb, c

Transform vectors:
fa = map fib a
fb = map fib b
c = zipWith plus fa fb

```
#i
#i
#i
#i
#include <ctime>
using namespace std;

int fib (int n)
{
   return (n < 2) ? n : fib(n-1) + fib(n-2);
}
int plus (int val1, int val2)
{
   return val1 + val2;
}
int main (int argc, char *argv[ ])
{
   const int factor = 23;
   const int vector_size = 40;
   vector<int> a, b, c;
   vector<int> fa, fb;
```

```
   cout << " Filling vectors..." << endl;
   1; i <= vector_size; i++)

   back(i % factor);
   back((vector_size + 1 - i) % factor);
   back(0);
   back(0);
   back(0);
   }

   cout << " Mapping..." << endl;
   transform(a.begin(),  a.end(),  fa.begin(), fib);
   cout << " Mapping..." << endl;
   transform(b.begin(),  b.end(),  fb.begin(), fib);
   cout << " Reducing..." << endl;
   transform(fa.begin(), fa.end(), fb.begin(), c.begin(),
      ::plus);

   cout << endl << " Result: (" ;
   ostream_iterator<int> output(cout, " ");
   copy(c.begin(), c.end(), output);
   cout << "\b)" << endl;

   return 0;
}
```

Map-Reduce (C++)

```
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
#include <ctime>
using namespace std;

int fib (int n)
{
   return (n < 2) ? n : fib(n-1) + fib(n-2);
}
int plus (int val1, int val2)
{
   return val1 + val2;
}
int main (int argc, char *argv[ ])
{
   const int factor = 23;
   const int vector_size = 40;
   vector<int> a, b, c;
   vector<int> fa, fb;

   cout << " Filling vectors..." << endl;
   for (int i = 1; i <= vector_size; i++)
   {
      a.push_back(i % factor);
      b.push_back((vector_size + 1 - i) % factor);
      c.push_back(0);
      fa.push_back(0);
      fb.push_back(0);
   }

   cout << " Mapping..." << endl;
   transform(a.begin(),  a.end(),  fa.begin(), fib);
   cout << " Mapping..." << endl;
   transform(b.begin(),  b.end(),  fb.begin(), fib);
   cout << " Reducing..." << endl;
   transform(fa.begin(), fa.end(), fb.begin(), c.begin(),
       ::plus);

   cout << endl << " Result: (" ;
   ostream_iterator<int> output(cout, " ");
   copy(c.begin(), c.end(), output);
   cout << "\b)" << endl;

   return 0;
}
```

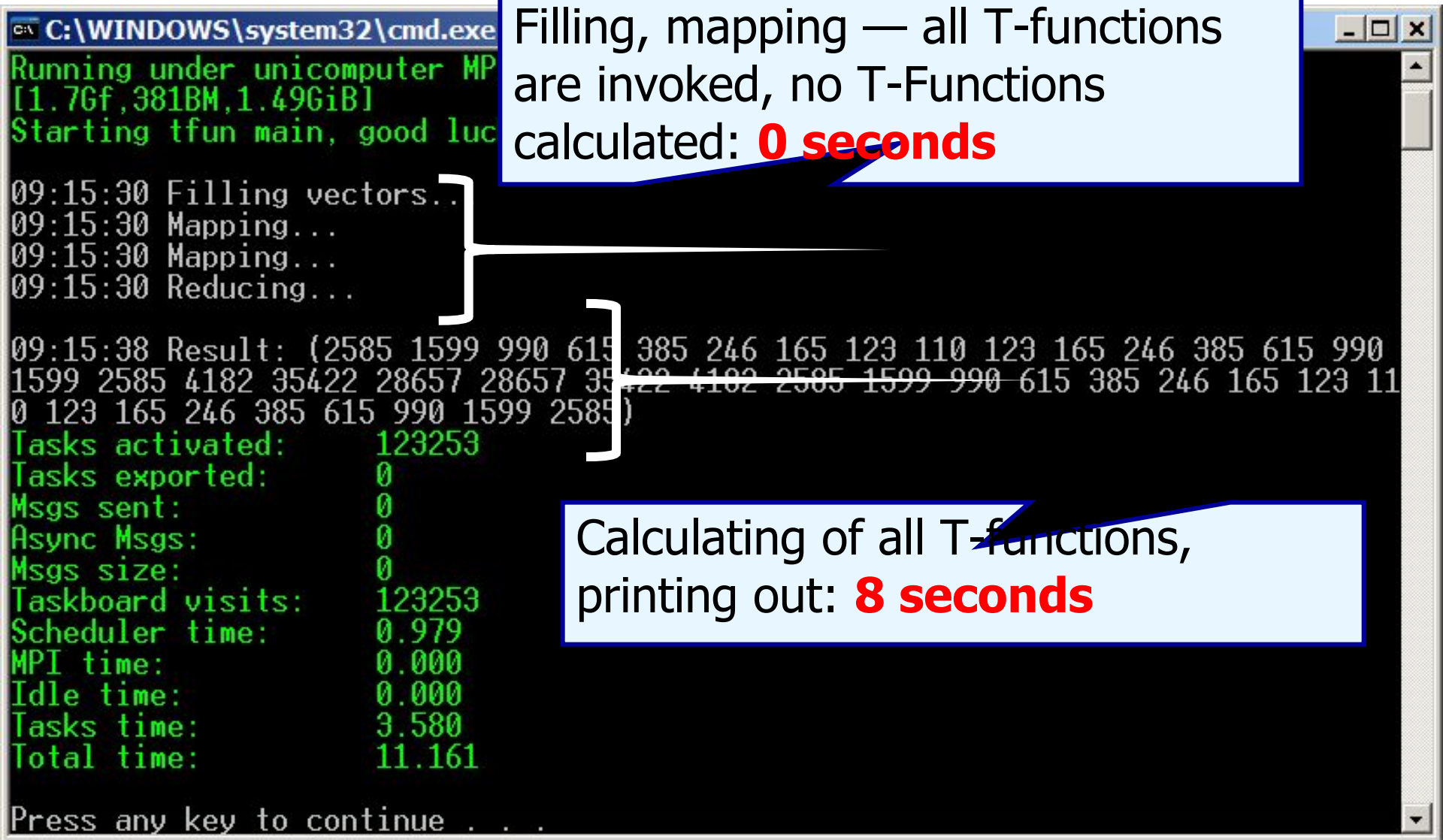
Map-Reduce (T++)

```
#include <vector>
#include <algorithm>
#include <functional>
#include <iostream>
#include <ctime>
using namespace std;

tfun int fib (int n)
{
   return (n < 2) ? n : fib(n-1) + fib(n-2);
}
tfun int plus (int val1, int val2)
{
   return val1 + val2;
}
tfun int main (int argc, char *argv[ ])
{
   const int factor = 23;
   const int vector_size = 40;
   vector<int> a, b, c;
   vector<tval int> fa, fb;
```

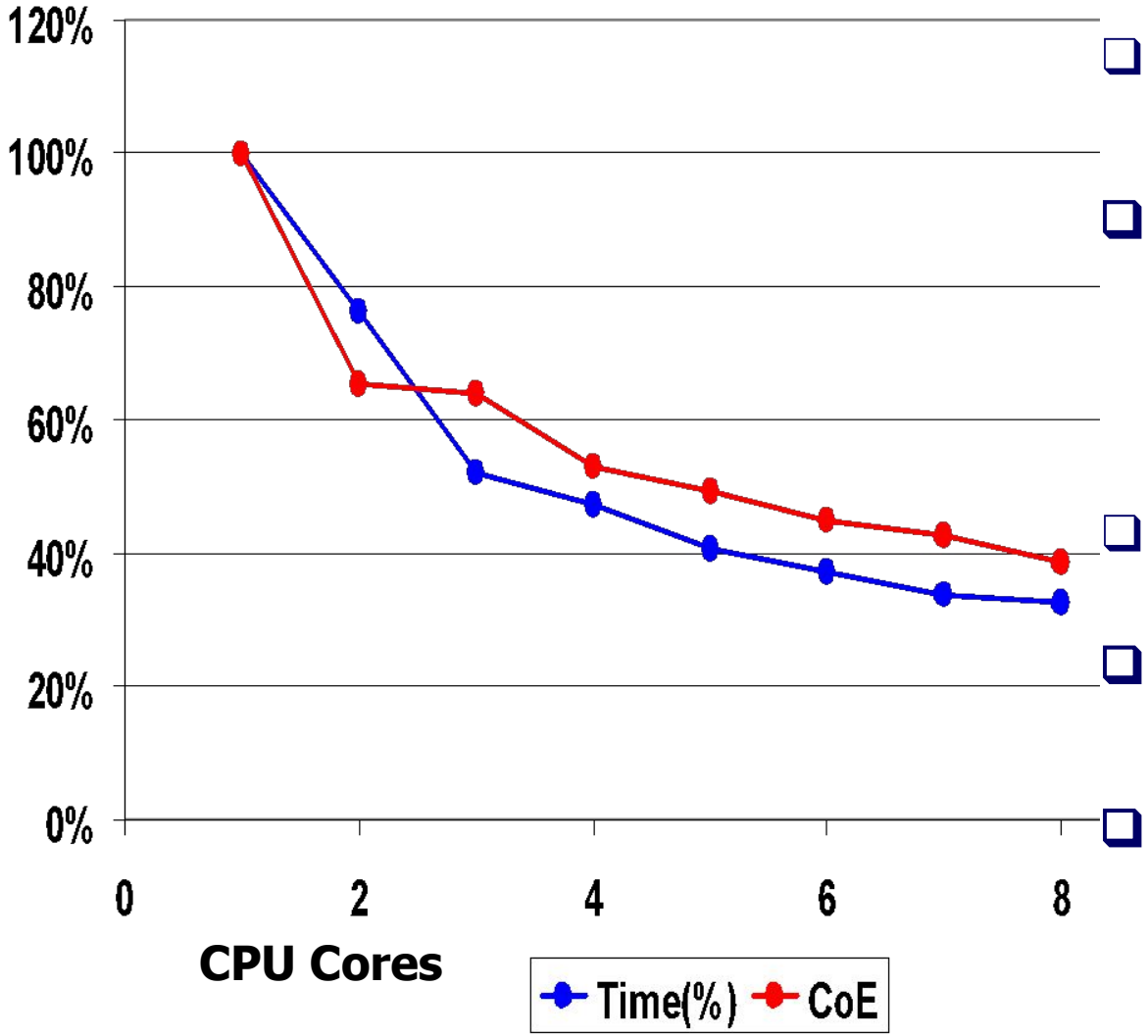
```
   cout << " Filling vectors..." << endl;
   for (int i = 1; i <= vector_size; i++)
   {
      a.push_back(i % factor);
      b.push_back((vector_size + 1 - i) % factor);
      c.push_back(0);
      fa.push_back(0);
      fb.push_back(0);
   }

   cout << " Mapping..." << endl;
   transform(a.begin(),  a.end(),  fa.begin(), fib);
   cout << " Mapping..." << endl;
   transform(b.begin(),  b.end(),  fb.begin(), fib);
   cout << " Reducing..." << endl;
   transform(fa.begin(), fa.end(), fb.begin(), c.begin(),
       ::plus);

   cout << endl << " Result: (" ;

   return 0;
}
```

Vector of T-values

Map-Reduce (T++): "Laziness"



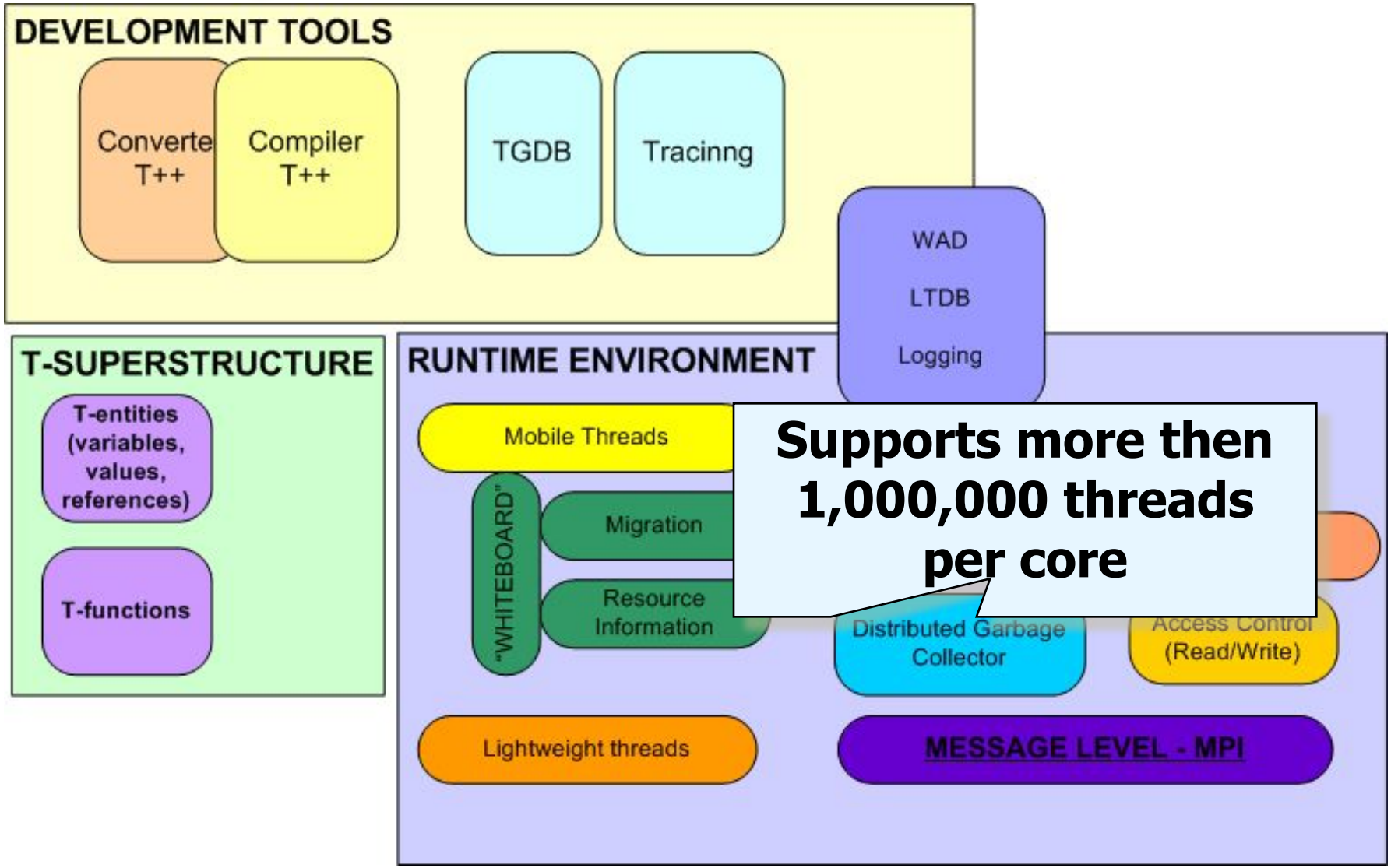
Map-Reduce (T++)



- WinCCS cluster, 4 nodes
- **CPU:** AMD Athlon 64 X2 Dual Core Processor 4400+ 2.21 GHz
- Gigabit Ethernet
- **time% =** $time_{tapp}(N)/timet_{app}(1)$
- **CoE =** $1/(n \times time\%)$

Inside OpenTS

Open TS: Environment



Supermemory

- Utilization: non-ready values, resource and status information, etc.
- Object-Oriented Distributed shared memory (OO DSM)
- Global address space
- DSM-cell versioning
- On top - automatic garbage collection

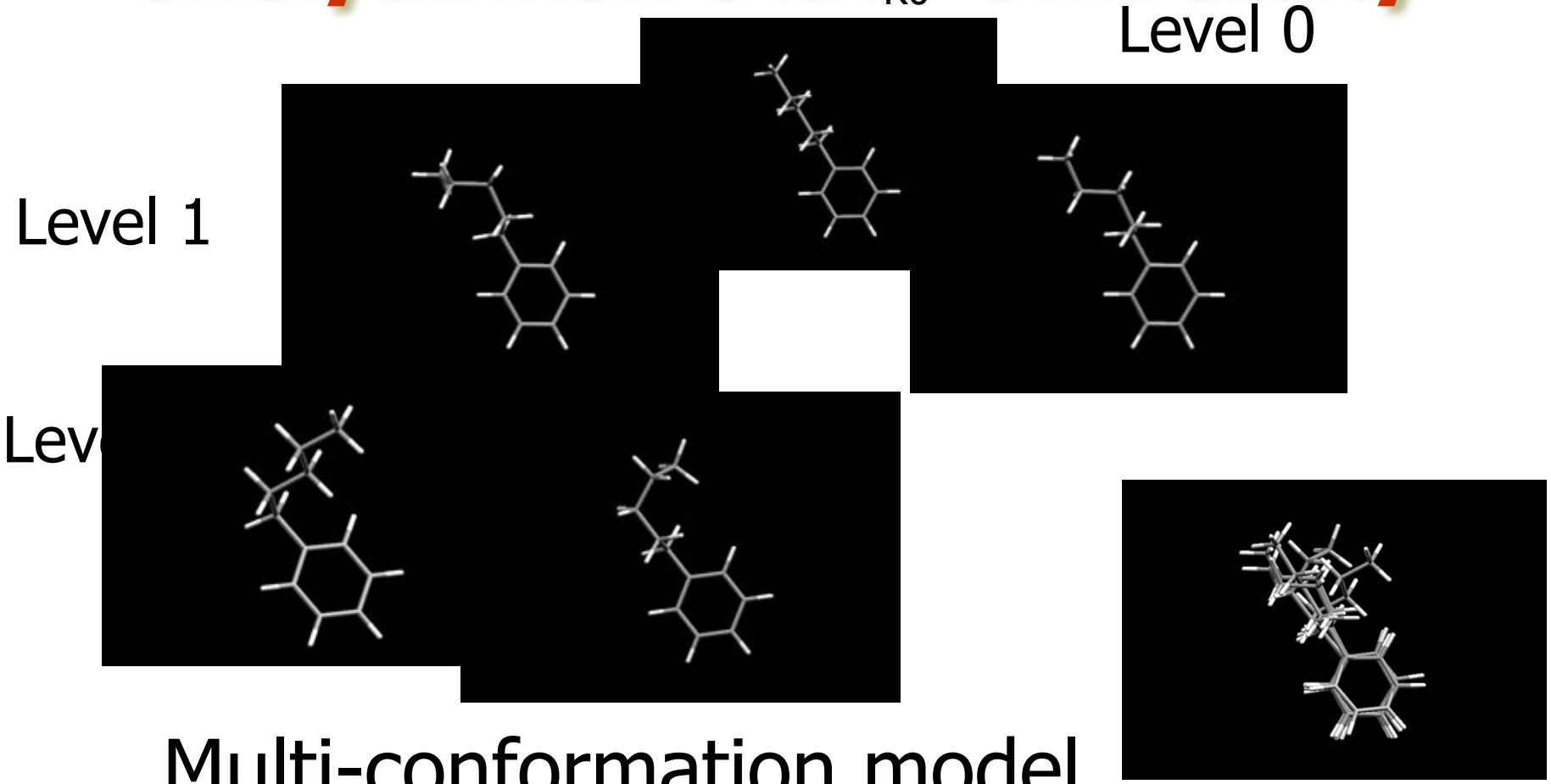
Multithreading & Communications

- **Lightweight threads**
  - PIXELS (1 000 000 threads)
- **Asynchronous** communications
  - A thread **"A"** asks non-ready value (or new job)
  - Asynchronous request sent: Active messages & Signals delivery over network to stimulate data transfer to the thread **"A"**
  - Context switches (including a quant for communications)
- **Latency Hiding** for node-node exchange

Open TS applications (selected)

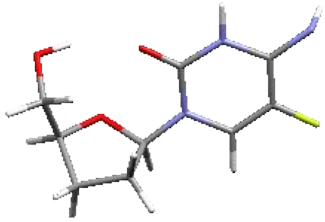
MultiGen

Chelyabinsk State University

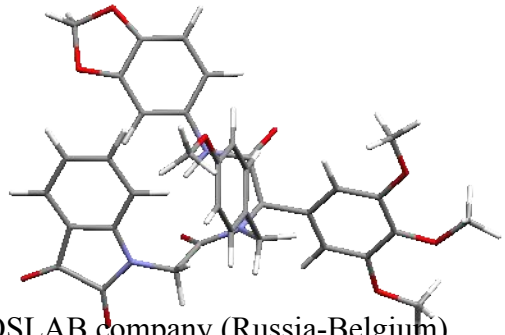


Multi-conformation model

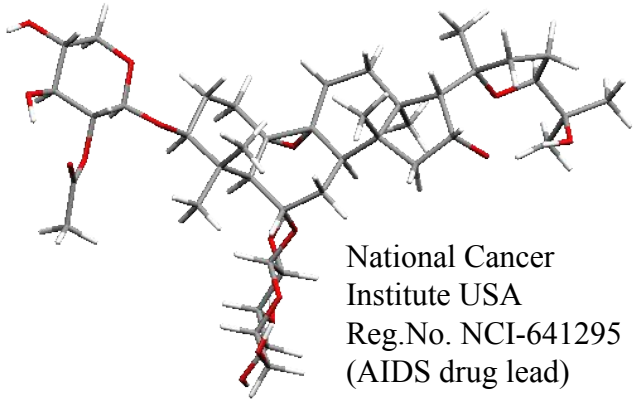
MultiGen: Speedup



National Cancer Institute USA
Reg.No. NCI-609067
(AIDS drug lead)



TOSLAB company (Russia-Belgium)
Reg.No. TOSLAB A2-0261
(antiphlogistic drug lead)



National Cancer
Institute USA
Reg.No. NCI-641295
(AIDS drug lead)

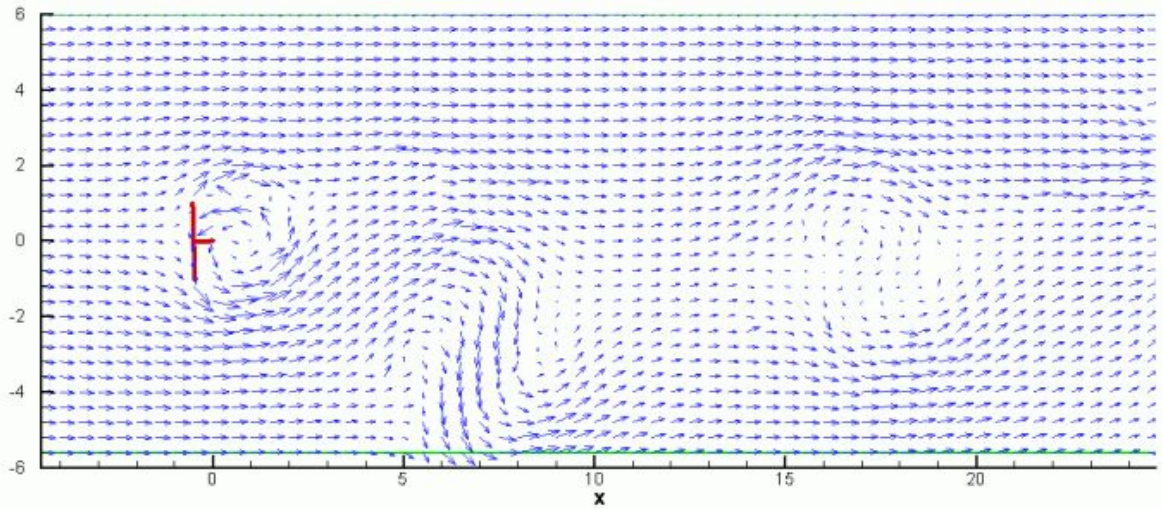
Substance	Atom number	Rotations number	Conformers	Execution time (min.:c)		
				1 node	4 nodes	16 nodes
NCI-609067	28	4	13	**9:33**	**3:21**	**1:22**
TOSLAB A2-0261	82	18	49	**115:27**	**39:23**	**16:09**
NCI-641295	126	25	74	**266:19**	**95:57**	**34:48**

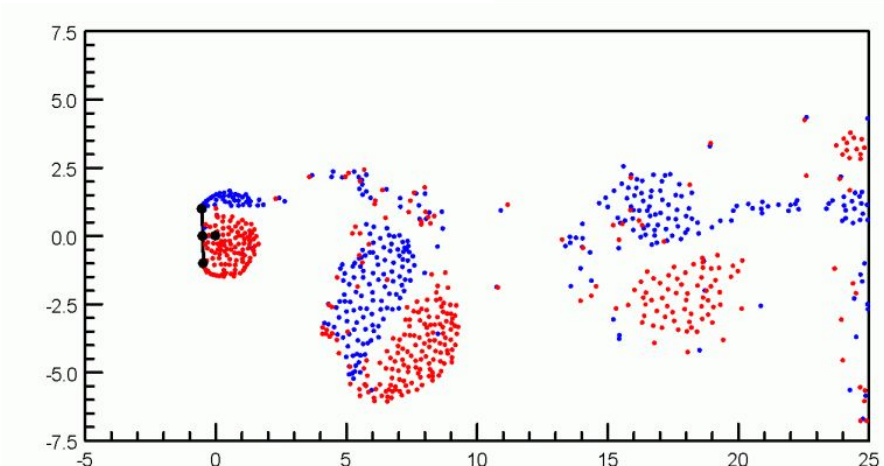
Aeromechanics

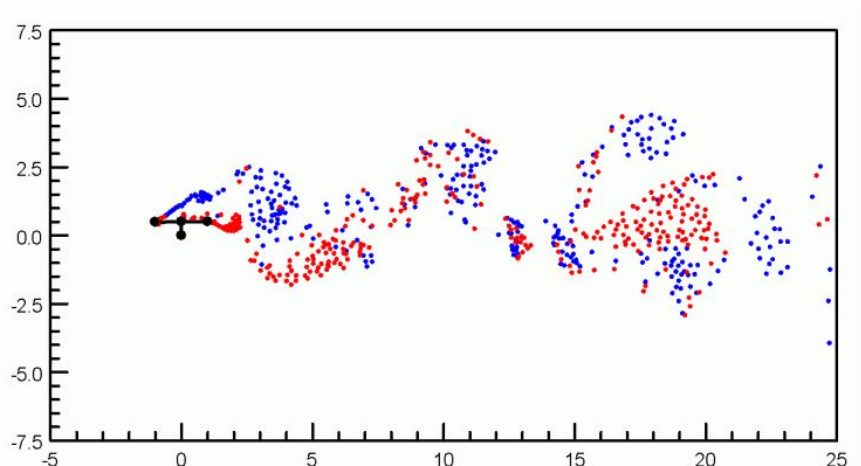
Institute of Mechanics, MSU

Belocerkovski's approach

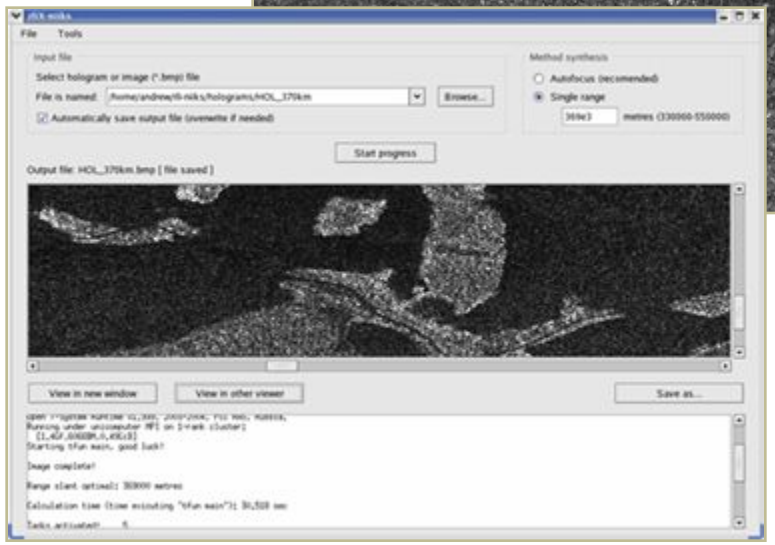
flow presented as a collection of small elementary whirlwind (colours: **clockwise** and **contra-clockwise** rotation)

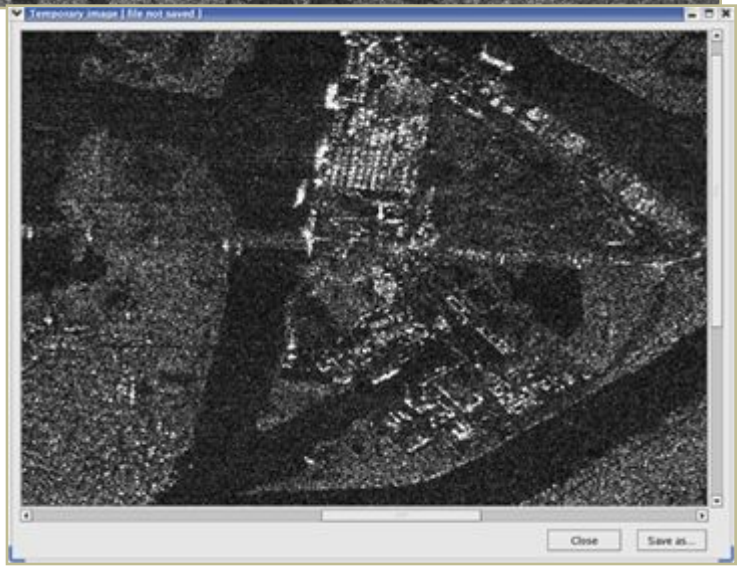






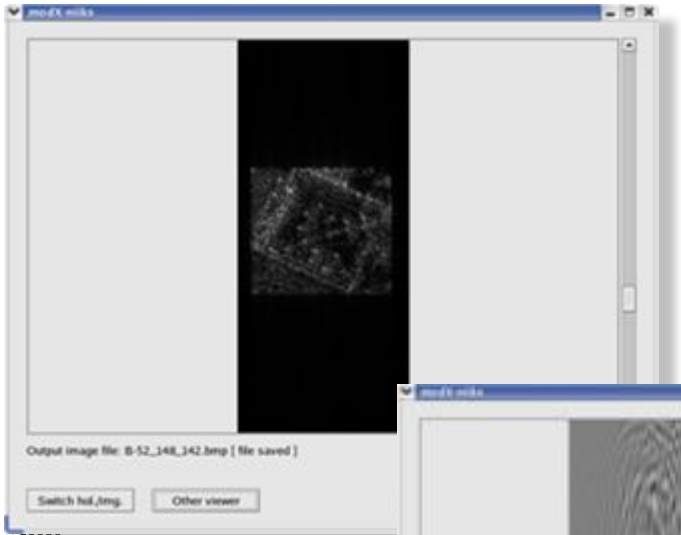
Creating space-born radar image from hologram

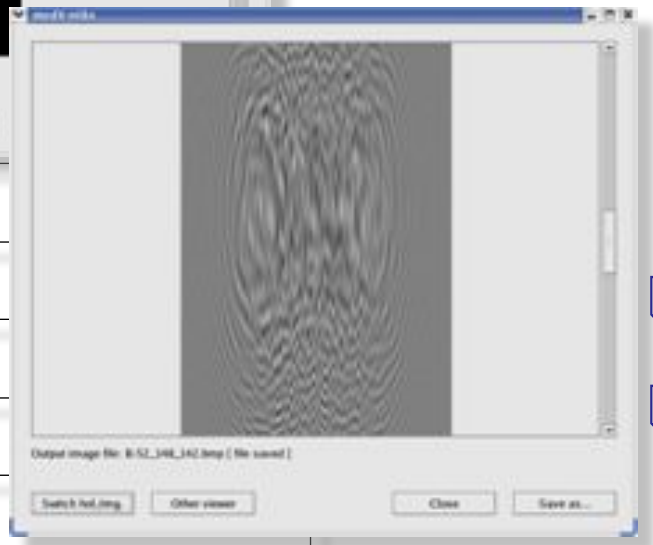


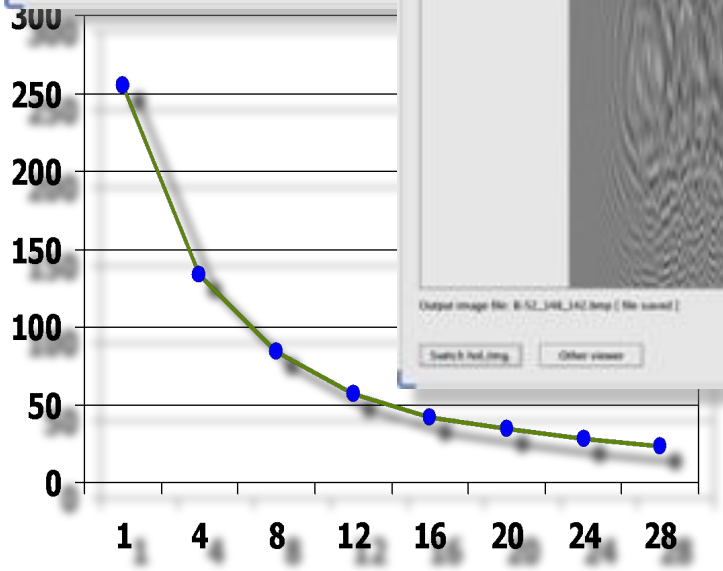


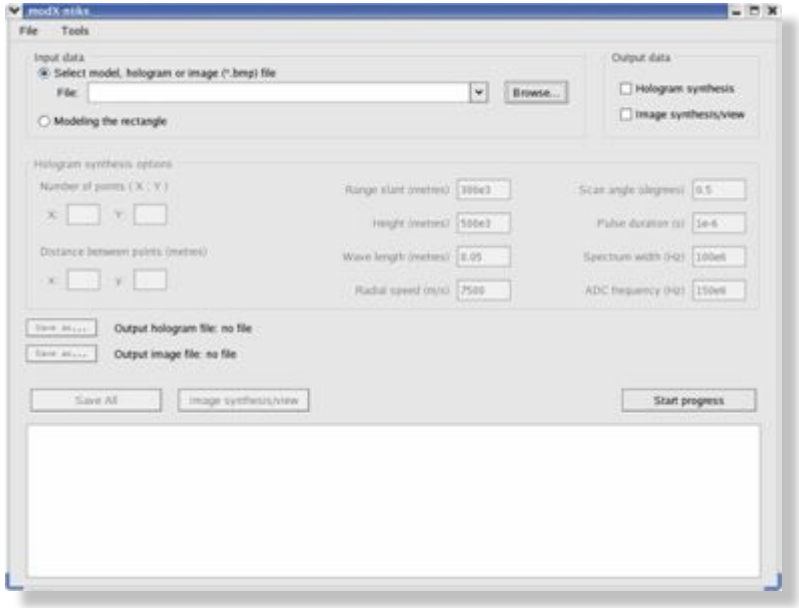
Space Research Institute Development

Simulating broadband radar signal





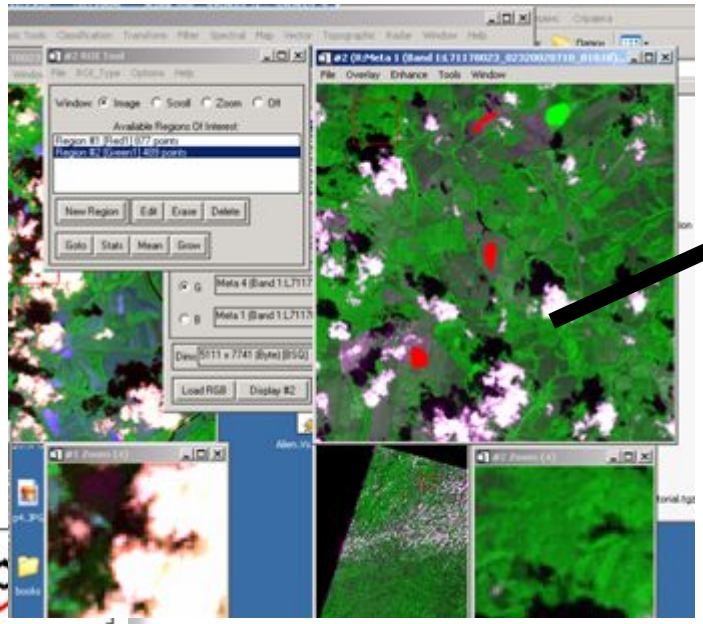


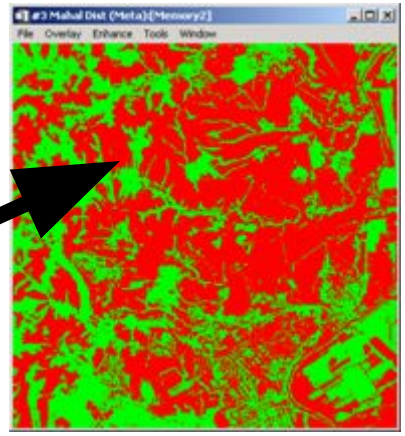


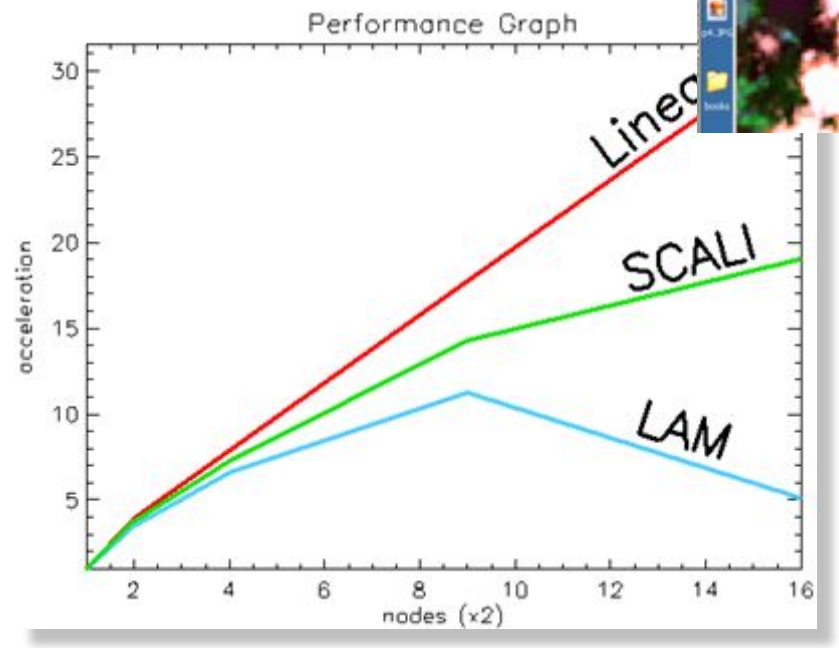
- Graphical User Interface
- Non-PSI RAS development team (Space research institute of Khrunichev corp.)

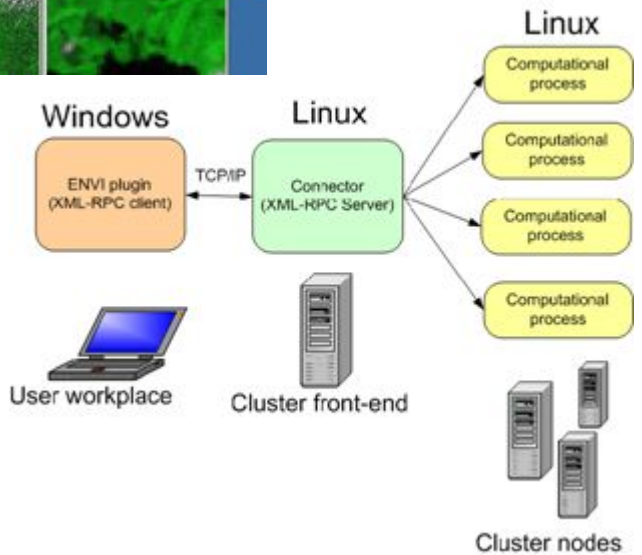
Landsat Image Classification

- Computational "web-service"









Open TS vs. MPI case study

Applications

- Popular and widely used
- Developed by independent teams (MPI experts)

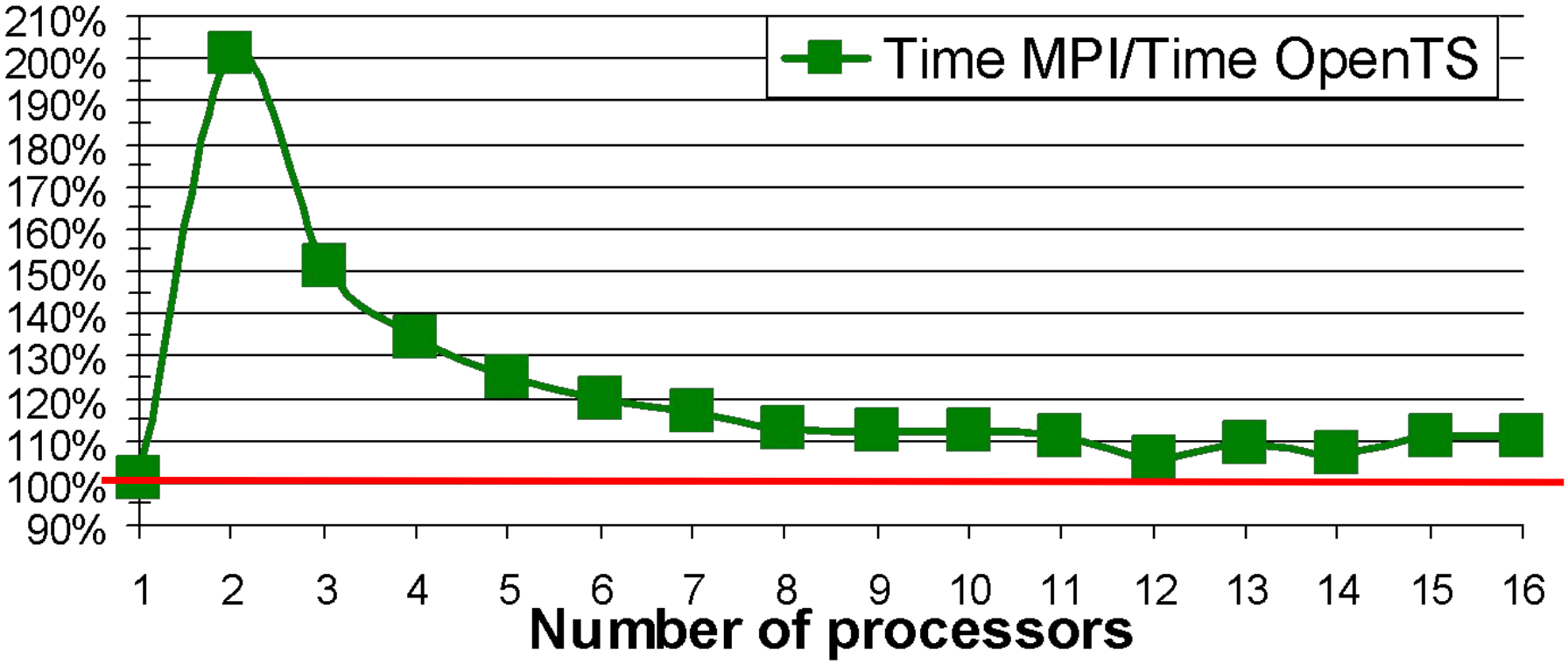
- **PovRay** – Persistence of Vision Ray-tracer, enabled for parallel run by a patch
- **ALCMD/MP_lite** – molecular dynamics package (Ames Lab)

T-PovRay vs. MPI PovRay: code complexity

Program	Source code volume
MPI modules for PovRay 3.10g	1,500 lines
MPI patch for PovRay 3.50c	3,000 lines
T++ modules (for both versions 3.10g & 3.50c)	200 lines

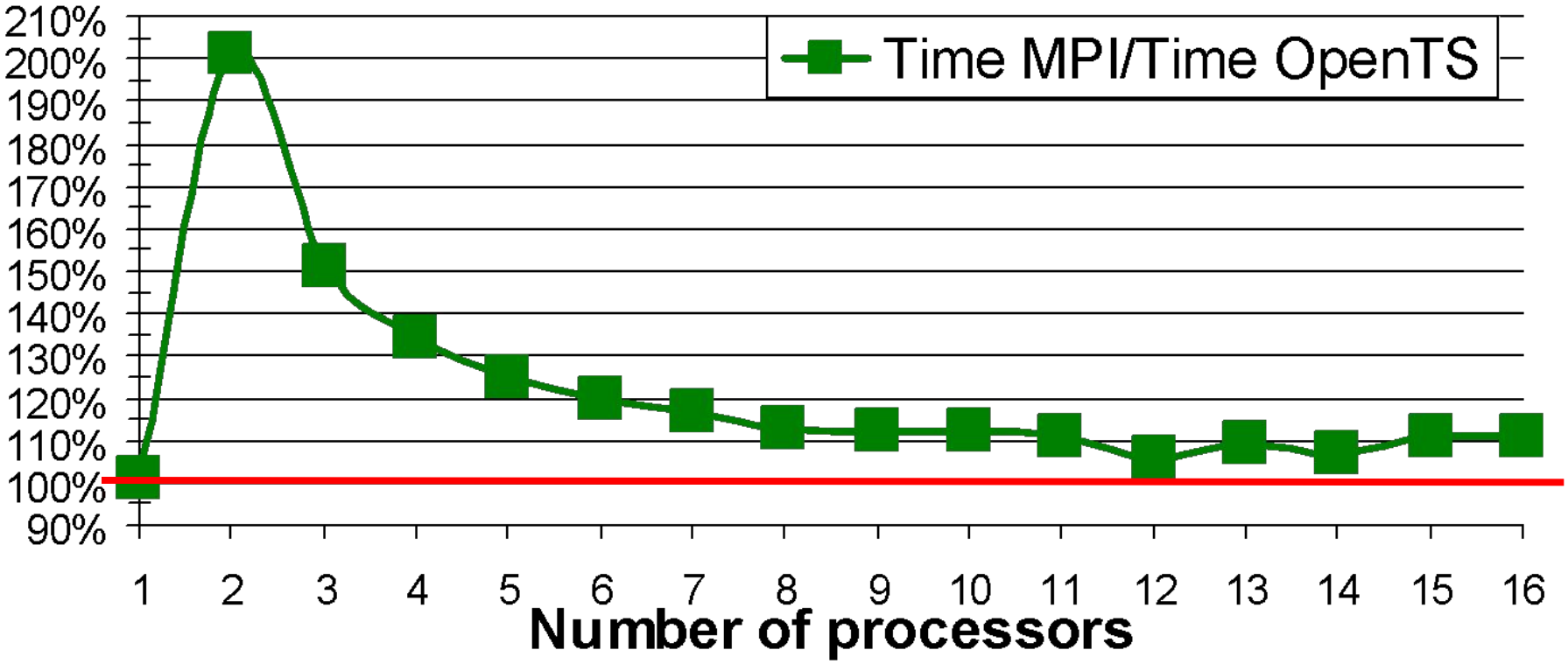
~7—15 times

T-PovRay vs. MPI PovRay: performance



2CPUs AMD Opteron 248 2.2 GHz RAM 4GB, GigE, LAM 7.1.1

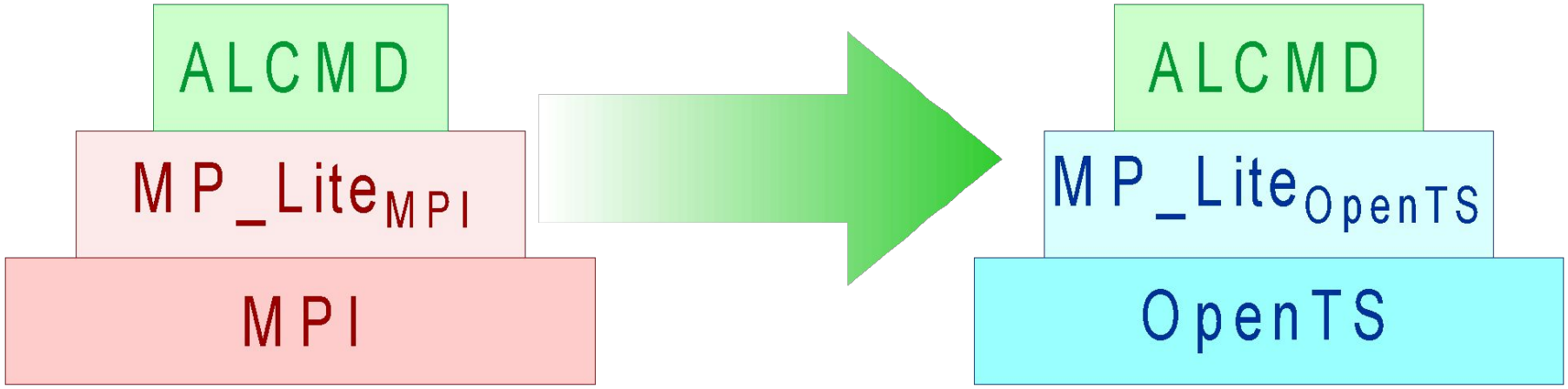
T-PovRay vs MPI PovRay: performance



2CPUs AMD Opteron 248 2.2 GHz RAM 4GB, GigE, LAM 7.1.1

ALCMD/MPI vs ALCMD/OpenTS

- MP_Lite component of ALCMD rewritten in T++
- Fortran code is left intact

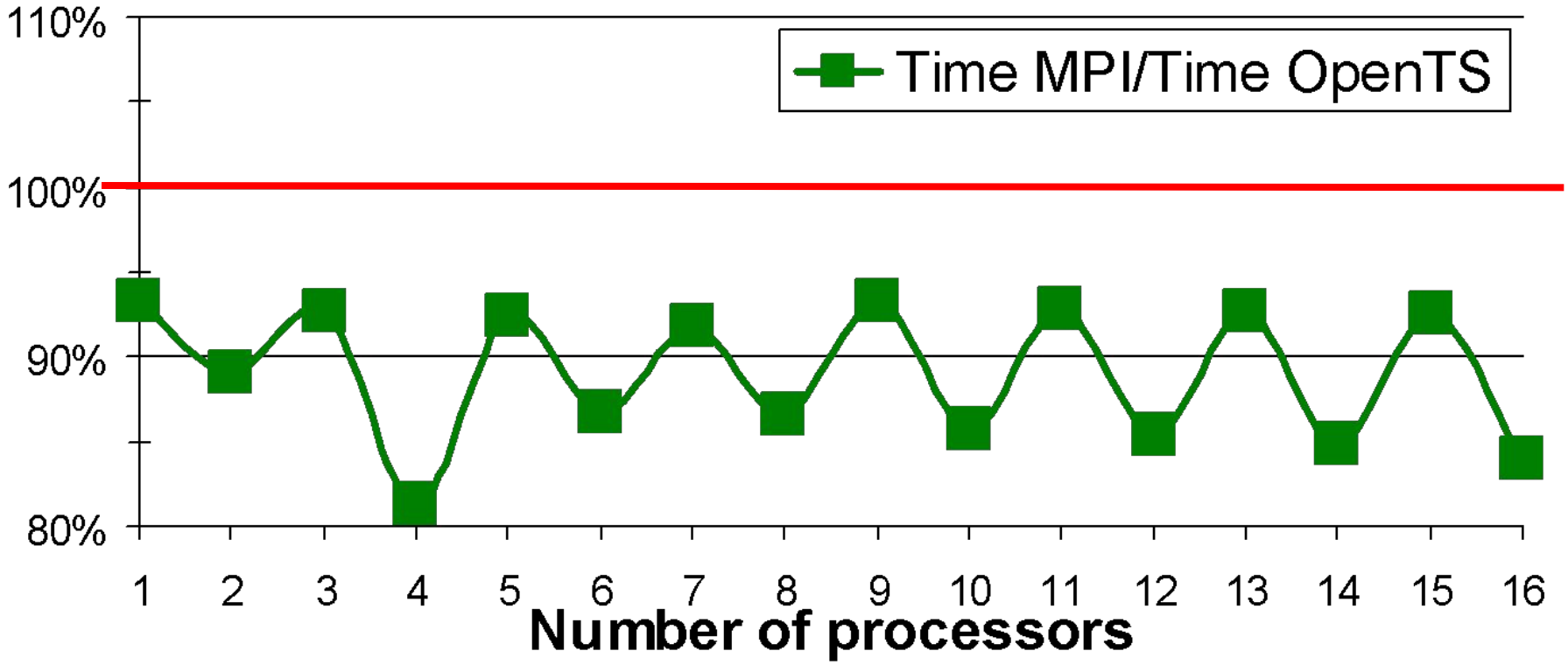


ALCMD/MPI vs ALCMD/OpenTS : code complexity

Program	Source code volume
MP_Lite total/MPI	~20,000 lines
MP_Lite,ALCMD-related/ MPI	~3,500 lines
MP_Lite,ALCMD-related/ OpenTS	500 lines

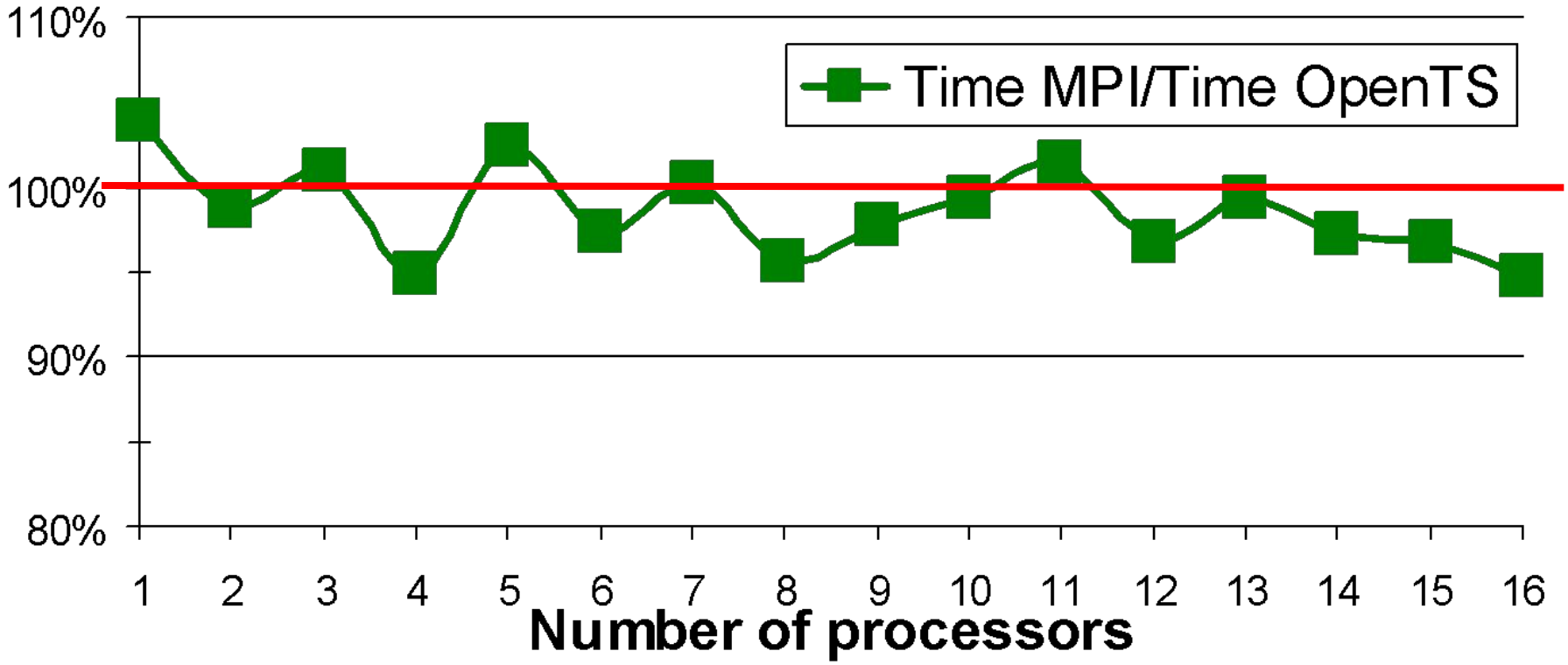
~7 times

ALCMD/MPI vs ALCMD/OpenTS: performance



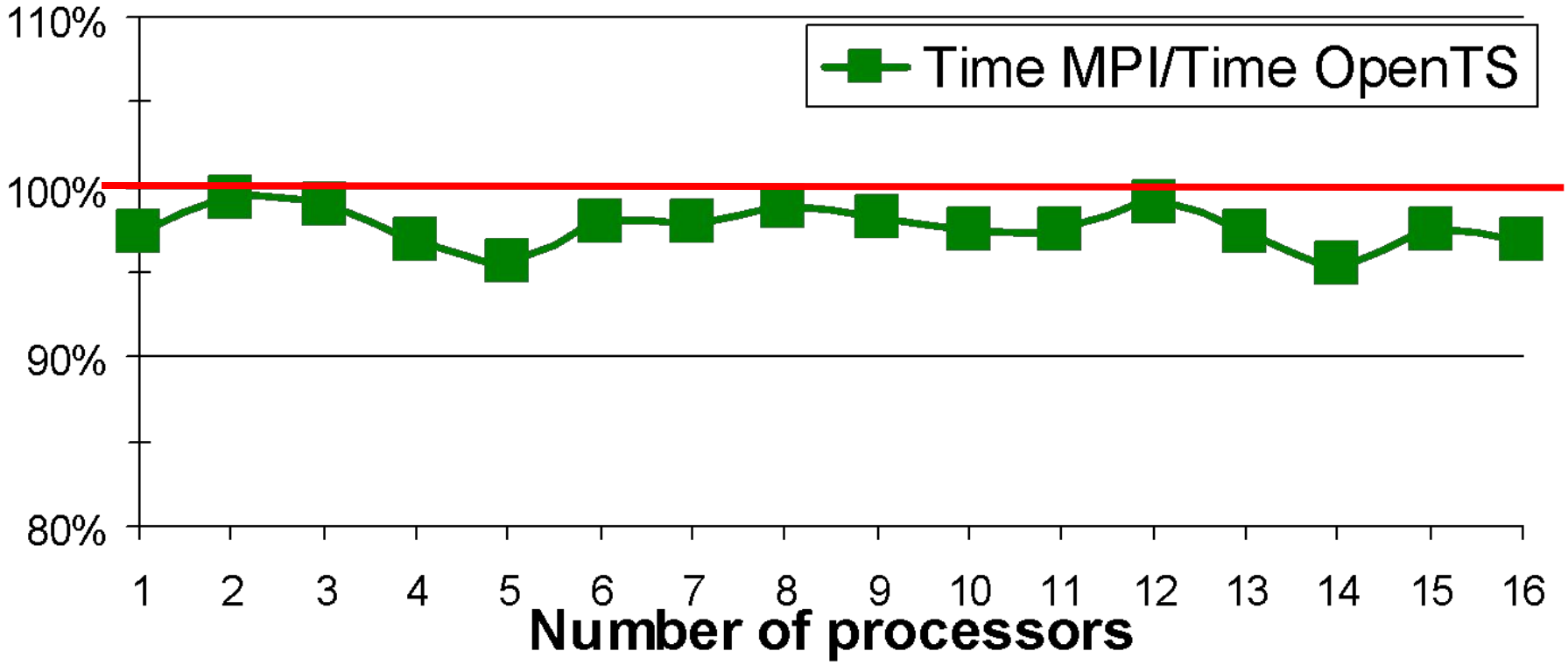
16 dual Athlon 1800, AMD Athlon MP 1800+ RAM 1GB, FastEthernet, LAM 7.0.6, Lennard-Jones MD, 512000 atoms

ALCMD/MPI vs ALCMD/OpenTS: performance



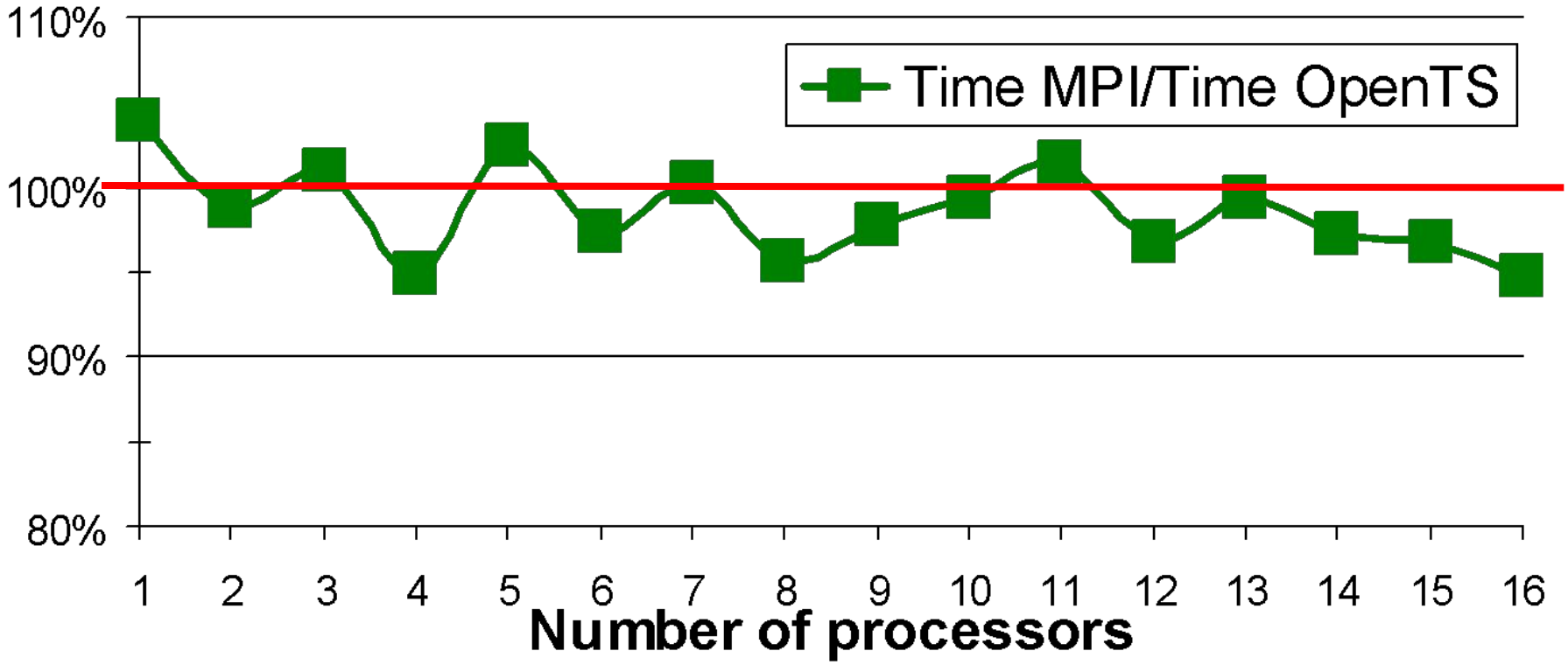
2CPUs AMD Opteron 248 2.2 GHz RAM 4GB,
GigE, LAM 7.1.1, Lennard-Jones MD, 512000 atoms

ALCMD/MPI vs ALCMD/OpenTS: performance



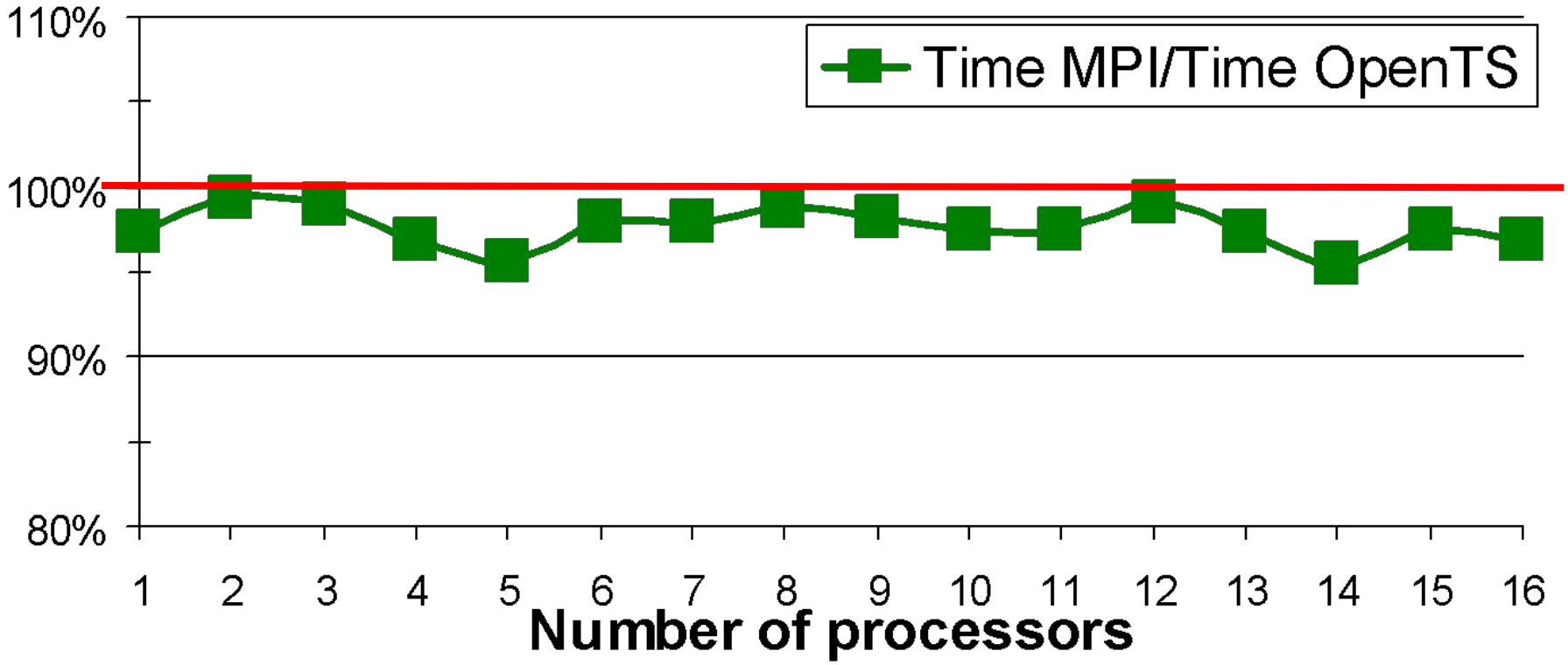
2CPUs AMD Opteron 248 2.2 GHz RAM 4GB,
InfiniBand,MVAMPICH 0.9.4, Lennard-Jones MD,512000 atoms

ALCMD/MPI vs ALCMD/OpenTS: performance



2CPUs AMD Opteron 248 2.2 GHz RAM 4GB,
GigE, LAM 7.1.1, Lennard-Jones MD, 512000 atoms

ALCMD/MPI vs ALCMD/OpenTS: performance



2CPUs AMD Opteron 248 2.2 GHz RAM 4GB,
InfiniBand,MVAMPICH 0.9.4, Lennard-Jones MD,512000 atoms

Porting OpenTS to MS Windows CCS

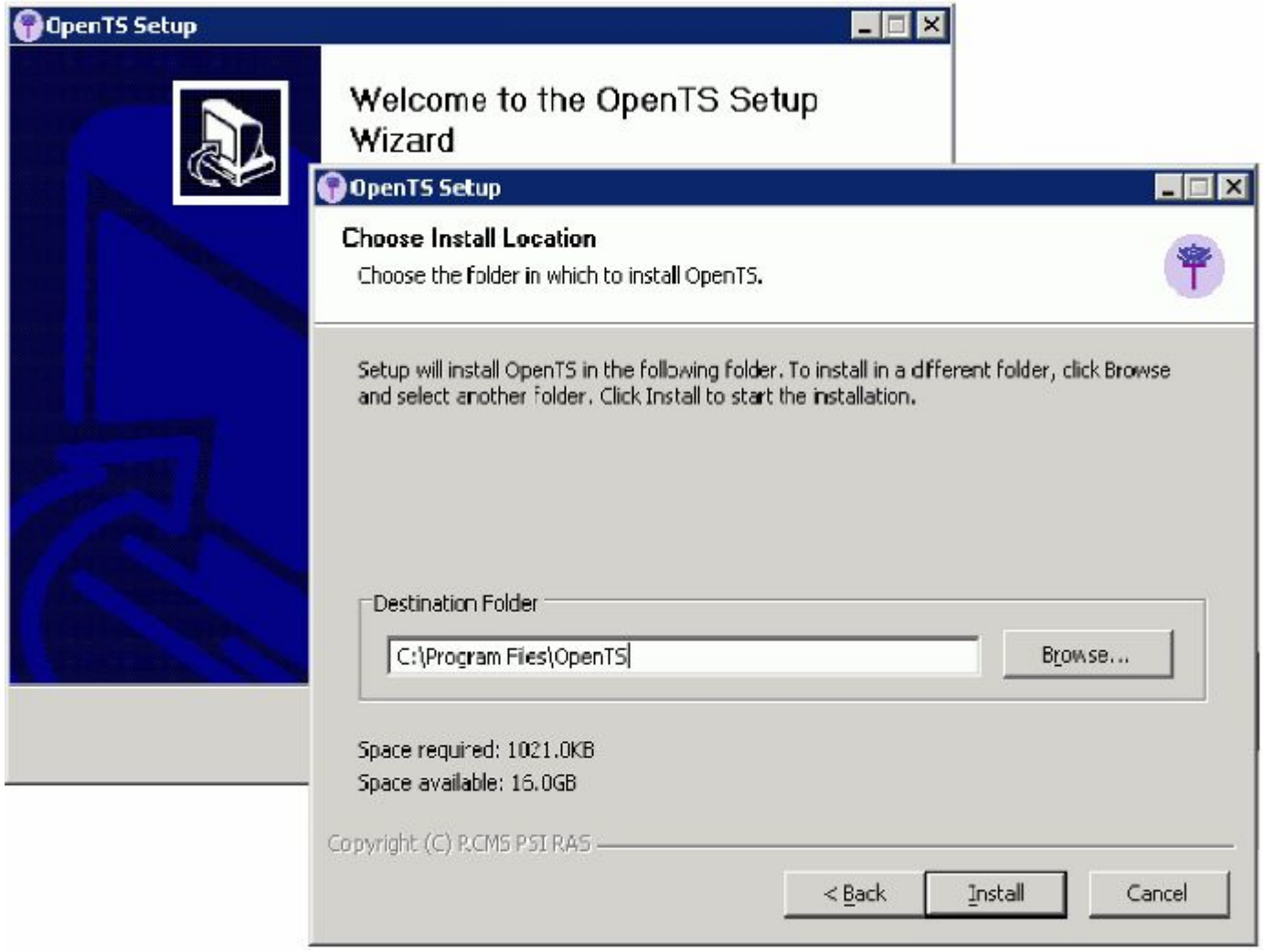
2006: contract with Microsoft "Porting OpenTS to Windows Compute Cluster Server"

- OpenTS@WinCCS
  - inherits all basic features of the original Linux version
  - is available under FreeBSD license
  - does not require any commercial compiler for T-program development — it's only enough to install VisualC++ 2005 Express Edition (available for free on Microsoft website) and PSDK

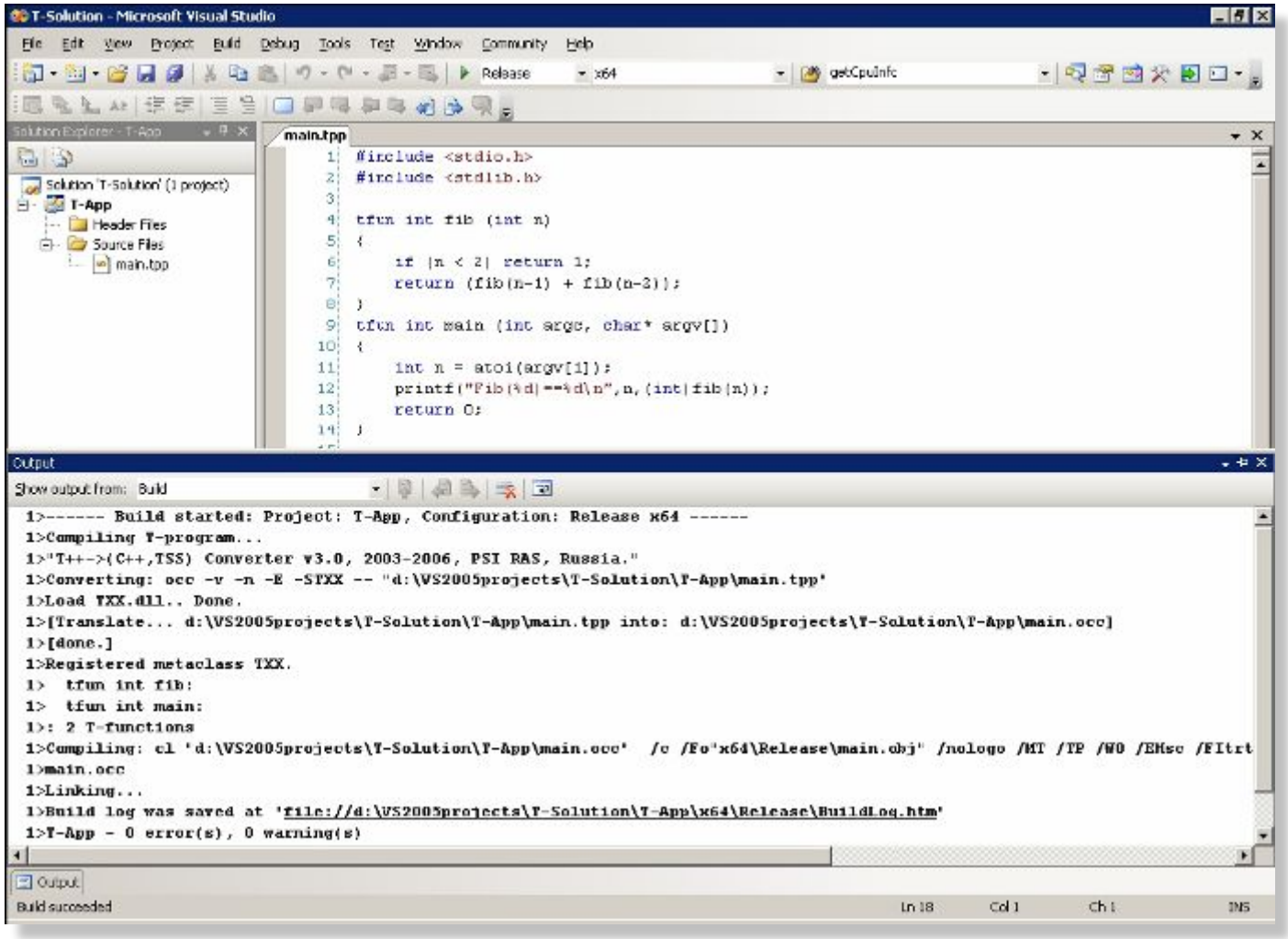
OpenTS@WinCCS

- AMD64 and x86 platforms are currently supported
- Integration into Microsoft Visual Studio 2005
- Two ways for building T-applications: command line and Visual Studio IDE
- An installer of OpenTS for Windows XP/2003/WCCS
  - Installation of WCCS SDK (including MS-MPI), if necessary
  - OpenTS self-testing procedure

Installer of OpenTS for Windows XP/2003/WCCS



OpenTS integration into Microsoft Visual Studio 2005



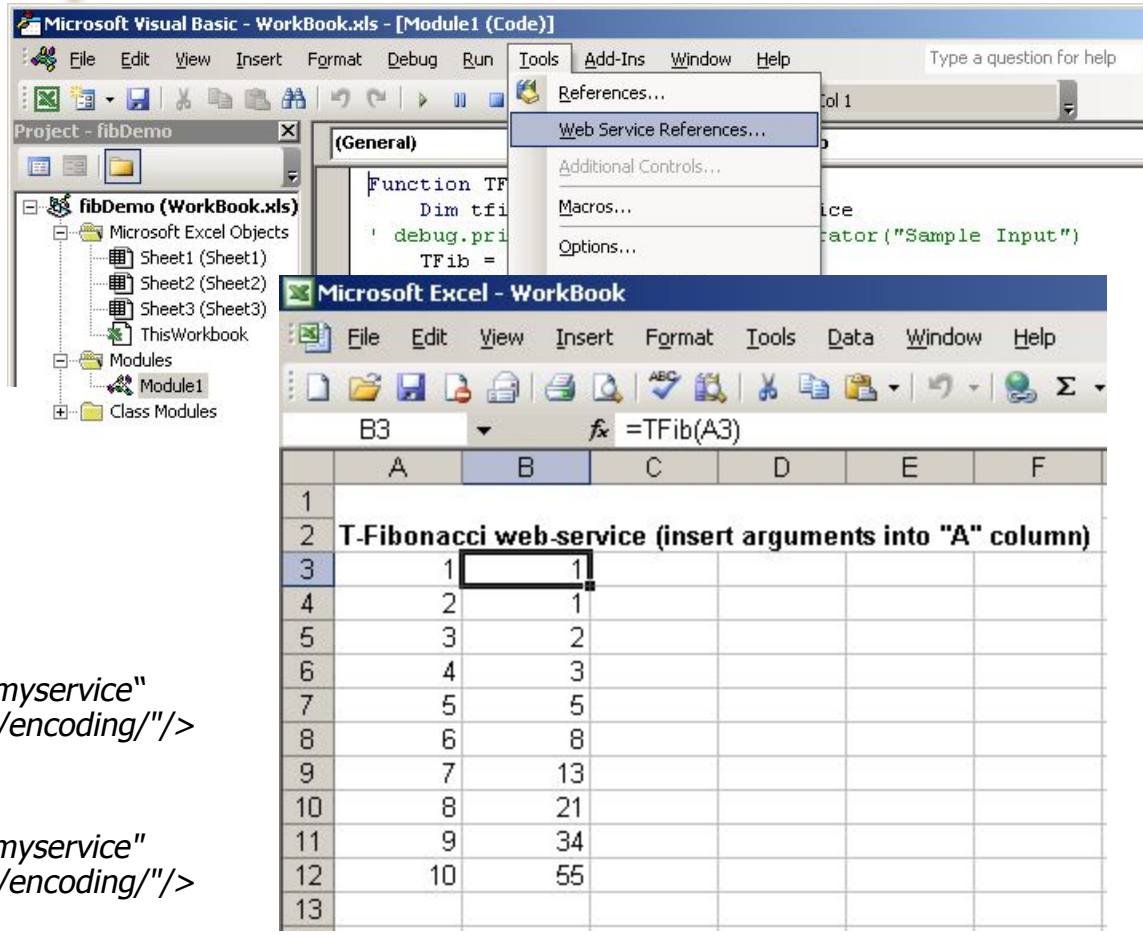
Open TS “Gadgets”

Web-services, Live documents

```
tfun int fib (int n) {
  return n < 2  ?  n :
        fib(n-1)+fib(n-2);
}
```

twsgen Perl script

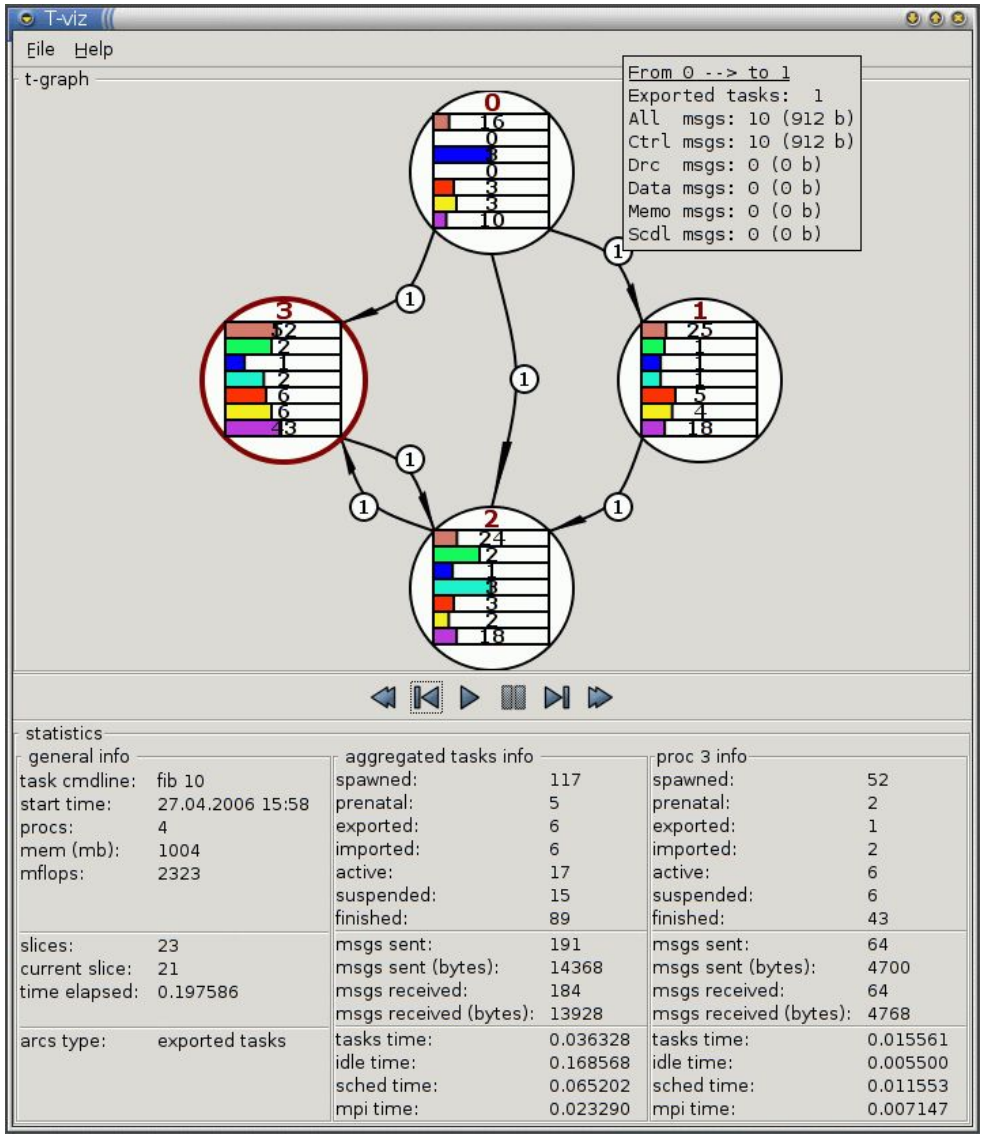
```
<operation name="wstfib">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:myservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:myservice"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
```



	A	B
3	1	1
4	2	1
5	3	2
6	4	3
7	5	5
8	6	8
9	7	13
10	8	21
11	9	34
12	10	55

Trace visualizer

- Collect trace of T-program execution
- Visualize performance metrics of OpenTS runtime



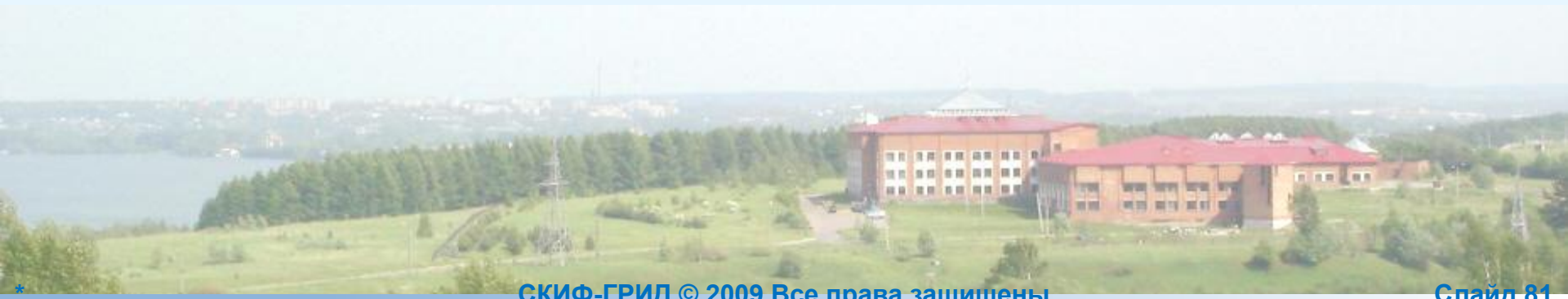
Fault-tolerance

- Recalculation based fault-tolerance
  - **(+)** Very simple (in comparison with full transactional model)
  - **(+)** Efficient (only minimal set of damaged functions are recalculated)
  - **(–)** Applicable only for functional programs
- Fault-tolerant communications needed (eg.: DMPI v1.0)
- Implemented (experimental version on Linux)

Some other Gadgets

- Other T-languages: T-Refal, T-Fortan
- Memoization
- Automatically choosing between call-style and fork-style of function invocation
- Checkpointing
- Heartbeat mechanism
- Flavours of data references: “normal”, “glue” and “magnetic” — lazy, eager and ultra-eager (speculative) data transfer

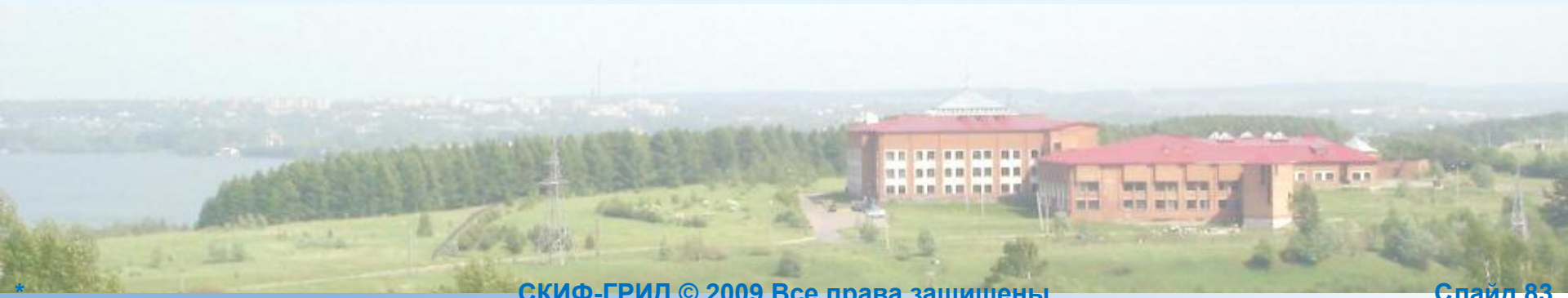
Full / Empty Bit (FEB) и Т-Система



FEB

- **Модель вычисления:**
  - «общая память»
  - легковесные нити
  - FEB — бит синхронизации на каждое слово
- **Тонкости:** фьючеры
- **Аппаратная реализация:** Cray XMT, MTA, T3D (2001 и далее)
- **Программная реализация: Sandia** National Laboratories: Qthreads — www.cs.sandia.gov/qthreads

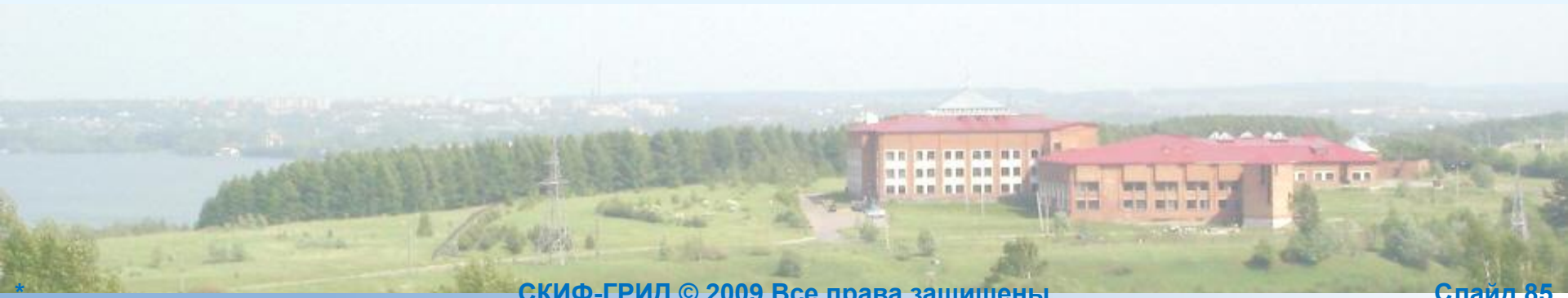
Монотонные объекты, как безопасное расширение функциональной модели



Идеи расширения

- Монотонные объекты — обладают свойством Черча-Россера
- Типичная жизнь монотонного объекта:
  - Создание и инициализация объекта
  - Поток обращений по обновлению («запись») и считыванию текущего состояния («чтение»)
  - Финализация и удаление объекта
- Примеры монотонных объектов (класса сумматоры)
- Примеры приложения с монотонным объектом
- Способы масштабируемой реализации монотонных объектов

Библиотеки односторонних обменов — SHMEM, Gasnet, ARMCI



Основные черты технологии SHMEM

- «Чужие» данные не обрабатываются на месте, а копируются предварительно туда, где они нужны — как в MPI
- Преобладающий режим копирования – запись, возможно, мелкозернистая
- Синхронизация – барьерная
- Важны
  - высокий message rate (например, для приложений со сложной организацией данных: неструктурных сеток и т.п.)
  - низкая подлинная латентность (для снижения размера зерна параллелизма, независимо от того, сложно или просто организованы данные)
- Однородность доступа не важна
- Message rate можно обеспечить программно, грамотной буферизацией
- Низкую подлинную латентность может дать только аппаратура

Технология SHMEM

- Рассчитана на полностью однородные многопроцессорные вычислители (общность системы команд, машинного представления чисел, одинаковая операционная система, один и тот же исполняемый файл)
- Программа на C, использующая SHMEM, должна включать файл заголовков shmem.h.
- **shmem_init()** — инициализация
- **my_pe()** — собственный номер процесса;
- **num_pes()** — число процессов

SHMEM: Односторонние обмены

- **put** --- односторонняя запись в чужую память
- **get** --- одностороннее чтение из чужой памяти
- Поддержаны передачи данных разных типов, одного значения, сплошного массива или массива, расположенного в памяти с шагом (например, столбец двумерного массива в C)
- **shmem_double_p(addr, value, pe)**
  shmem_float_put(addr, src, len, pe)
- Можно обмениваться областями памяти статических (но не автоматических!) переменных. И даже поддержан специальный **malloc**

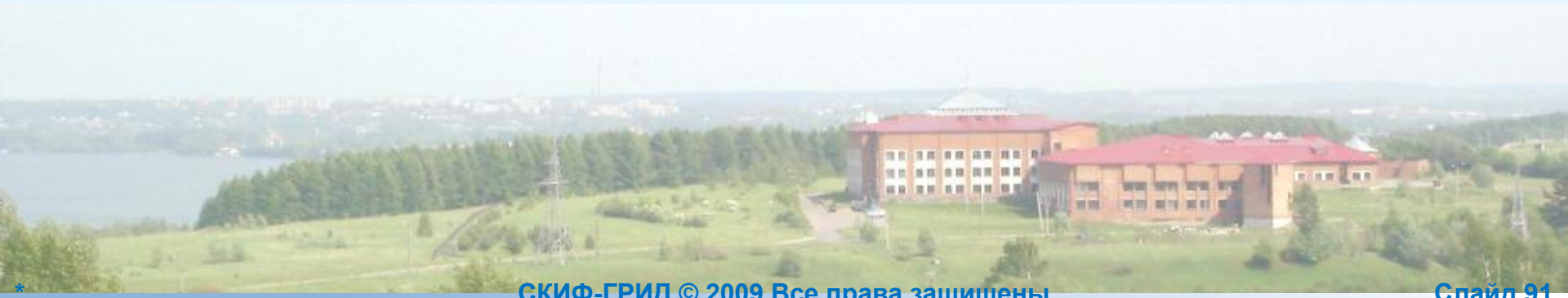
SHMEM: Операции синхронизации

- Возможность выполнить барьерную синхронизацию всех или лишь указанных процессов.
- При выполнении синхронизации гарантируется, что все выданные до барьера запросы типа **put** будут завершены.
- **shmem_barrier_all()**
- **shmem_barrier(start, stride, size, sync)**
  - **step = 2^{stride}**
  - синхронизуются процессы с номерами **start, strat+step, … strat+step*(size-1)**
  - **sync** — рабочий массив (расписать нулями!) типа **long**, длиной **_SHMEM_BARRIER_SYNC_SIZ**E

Ожидание переменной

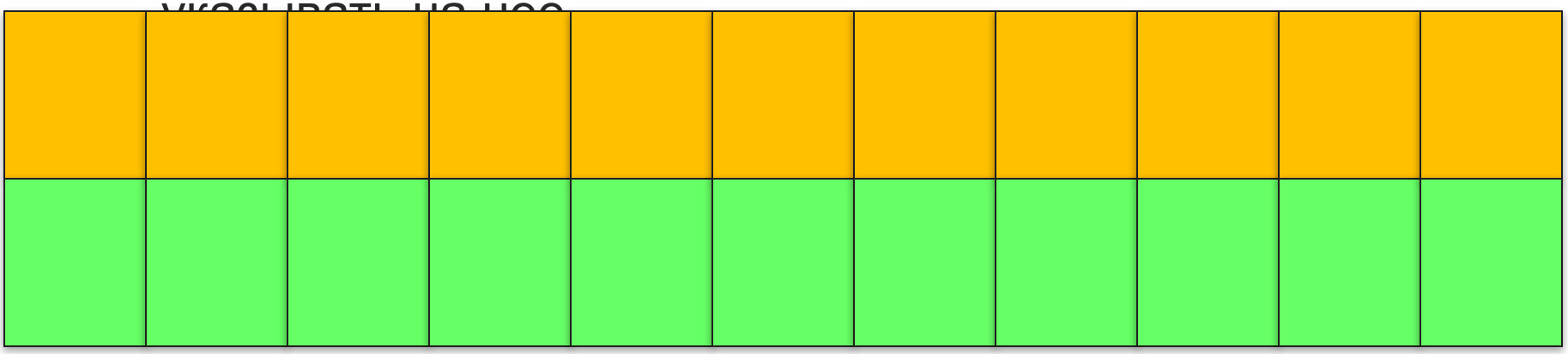
- **shmem_wait(&var, value)** — на «==»
  - ожидание **&var == value**
- **shmem_int_wait_until(&var, SHMEM_CMP_GT, value)**
  - ожидание **&var > value**
- **shmem_fence() —** гарантирует, что все выданные ранее из данного процесса запросы типа **put** будут завершены.

Модели PGAS (Partitioned Global Address Space), DSM (Distributed Shared Memory), языки Co-Array Fortran, UPC…



Модель памяти

- Разделяемая (shared) память
  - Любой процесс может использовать ее или указывать на нее
- Приватная память
  - Только «локальный» процесс может использовать ее или указывать на нее



Что такое UPC

- Unified Parallel C
- Расширение ANSI C примитивами задания явного параллелизма
- Основан на «distributed shared memory»
- Основные идеи
  - Сохранить философию C:
  - Программист умен и аккуратен
  - Близость к железу (насколько возможно) для лучшей эффективности (но можно получить и проблемы)
  - Простой и привычный синтаксис

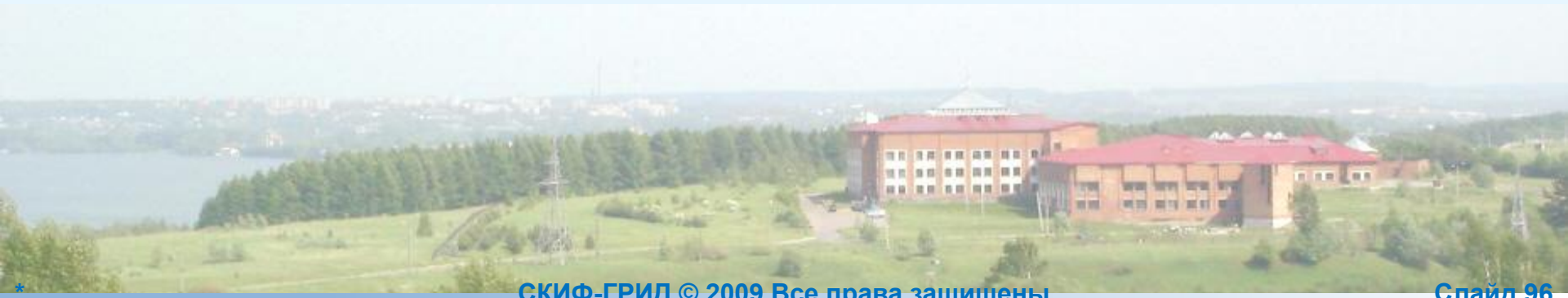
Модель исполнения

- Несколько процессов (нитей 0..THREADS-1) работают независимо
- MYTHREAD определяет номер процесса
- THREADS — число процессов
- Когда необходимо для синхронизации используются
  - Барьеры
  - Блокировки
  - Управление поведением памяти

Пример 1

```
//vect_add.c
#include <upc_relaxed.h>
#define N 100*THREADS
shared int a[N], b[N], c[N];
void main(){
    int i;
    for(i=0; i<N; i++)
    if (MYTHREAD==i%THREADS)
        c[i]=a[i]+b[i];
}
```

Вместо заключения



Спасибо за внимание!

? — !