Additional Parallelization of Existing MPI Programs Using SAPFOR

Nikita Kataev and Alexander Kolganov

Keldysh Institute of Applied Mathematics RAS, Moscow, Russia

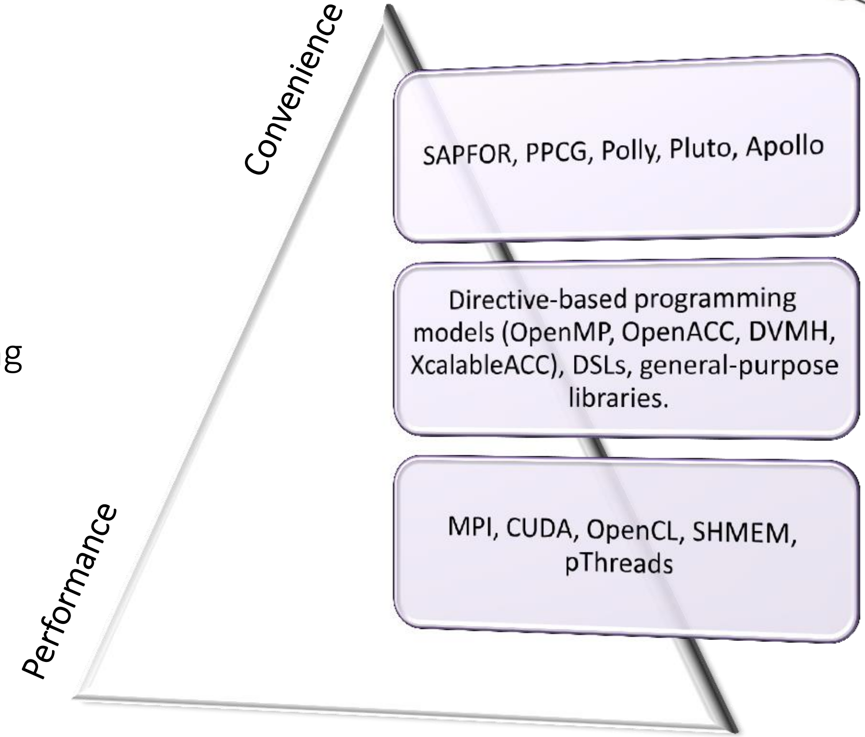
September 14, 2021 | Kaliningrad

Parallel Programming Tools

Automatic parallelizing compilers return a fully parallelized source code (maybe not optimal) for a given sequential one.

Directive-based models simplify programming and increase software maintainability while still providing high performance.

Low-level models, give programmers fine-grained control over the program execution and allow them to gain the best performance.


Convenience
Performance
SAPFOR, PPCG, Polly, Pluto, Apollo
Directive-based programming models (OpenMP, OpenACC, DVMH, XcalableACC), DSLs, general-purpose libraries.
MPI, CUDA, OpenCL, SHMEM, pThreads

Our approach to the exploitation of intra-node parallelism in existing MPI programs (C and Fortran):

- a directive-based programming model (DVMH),
- automation tools (SAPFOR),
- user participation.

DVMH Programming Model

Development of high-level parallel programming languages.
Directive-based programming models.

CUDA

OpenMP

MPI

DVMH

- Directive-based programming model which aims to create parallel programs for heterogeneous computational clusters (GPU NVidia, Intel Xeon Phi, multicore CPUs).
- The model includes two programming languages which are the extensions of standard C and Fortran languages by parallelism specifications: *CDVMH* and *Fortran-DVMH*
- The parallel program is developed in terms of a sequential one.

Fortran-DVMH Program with Data Distribution

```
program jacoby_dvmh
  parameter (l=4096, itmax=100)
  real a(l,l), b(l,l), eps
!DVM$ DISTRIBUTE(BLOCK, BLOCK) :: A
!DVM$ ALIGN B(I,J) WITH A(I,J)
  ...
  do it = 1, itmax
    eps = 0.
!DVM$ REGION
!DVM$ PARALLEL (J,I) ON  A(I,  J), REDUCTION(MAX(EPS))
    do  j = 2, l-1
      do  i = 2, l-1
        eps = max(eps, abs(B(i, j) - A(i, j)))
        a(i, j) = b(i, j)
      enddo
    enddo
!DVM$ PARALLEL (J,I) ON B(I,  J), SHADOW_RENEW(A)
    do  j = 2, l-1
      do  i = 2, l-1
        b(i, j) = (a(i-1, j) + a(i, j-1) + a(i+1, j) + a(i, j+1)) / 4
      enddo
    enddo
!DVM$ END REGION
  enddo
!DVM$ GET_ACTUAL(B)
  print *, b
end
```

Main directives:

- data distribution,
- compute regions and specifications of CPU-to-GPU data transfer,
- computation distribution,
- variable properties and remote data.

Fortran-DVMH Program without Data Distribution

```
program jacoby_dvmh
  parameter (l=4096, itmax=100)
  real a(l,l), b(l,l), eps


  ...
  do it = 1, itmax
    eps = 0.
!DVM$ REGION
!DVM$ PARALLEL (J,I), REDUCTION(MAX(EPS)), TIE(A(I,J), B(I, J))
    do  j = 2, l-1
      do  i = 2, l-1
        eps = max(eps, abs(B(i, j) - A(i, j)))
        a(i, j) = b(i, j)
      enddo
    enddo
!DVM$ PARALLEL (J,I), TIE(A(I,J), B(I, J))
    do  j = 2, l-1
      do  i = 2, l-1
        b(i, j) = (a(i-1, j) + a(i, j-1) + a(i+1, j) + a(i, j+1)) / 4
      enddo
    enddo
!DVM$ END REGION
  enddo
!DVM$ GET_ACTUAL(B)
  print *, b
end
```

Main directives:

- compute regions and specifications of CPU-to-GPU data transfer,
- computation distribution,
- variable properties and array access patterns.

DVMH-based Parallelization of MPI Programs

The DVMH runtime system does not participate in any inter-processor interaction, and it works locally in each MPI process.

The **tie** specification is used to set correspondence between loops in the parallel loop nest and dimensions of the arrays.

The capabilities of the DVM system allows us:

- to use parallelism on shared memory with using CPU cores (OpenMP threads) or to use graphics accelerators,
- to perform the automatic data transformation on GPUs, and to use simplified management of data movements between CPU and GPUs memories,
- to select optimization parameters of DVMH runtime system,
- to use tools for debugging and performance analyzing of parallel programs.

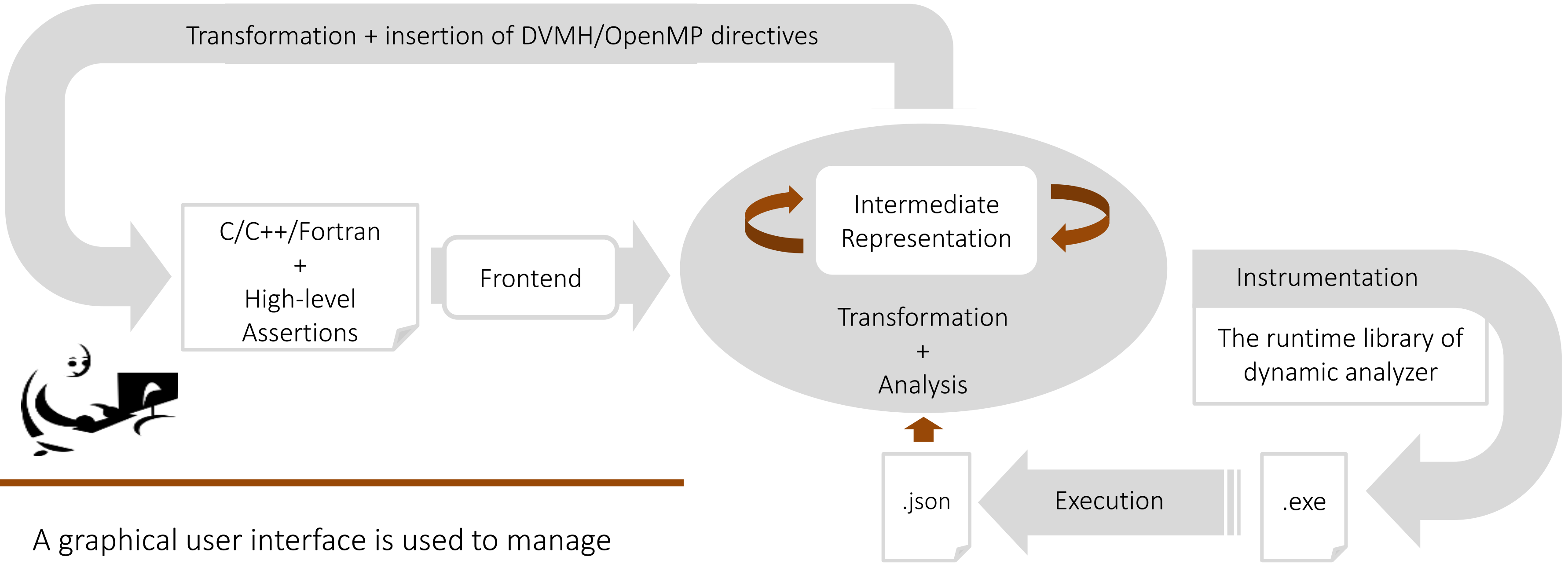
Goals of SAPFOR

SAPFOR (System For Automate Parallelization) is a software development suit that is focused on cost reduction of manual program parallelization.

The main goals of SAPFOR development:

- Exploration of sequential programs (program analysis and profiling).
- Automatic parallelization (according to DVMH or OpenMP models) of a well-formed program for which a programmer maximizes algorithm-level parallelism and asserts high-level properties (implicit parallel programming methodology).
- Semi-automatic program transformation to obtain a well-formed sequential version of the original program.

Architecture of SAPFOR



- A graphical user interface is used to manage parallelization.
- Build automation tools, such as Make, can be also used to run program analysis.

Automatic Parallelization Using DVMH Model

Parallelization for compute devices with shared memory (multi-core CPU or accelerator) requires that three kinds of annotations be inserted into the source code:

- specifications of the loops which can be executed in parallel, as well as specifications of private and reduction variables, and a pattern of array accesses,
- specification of the compute regions which can be executed on the accelerators, each region may enclose one or more parallel loops,
- high-level specifications of data transfer between a memory of CPU and a memory of accelerator (actualization directives).

For each loop the following constraints are examined:

- safety of control flow (the absence of I/O operations, side effects, etc.),
- safety of memory accesses (absence of loop-carried data dependencies and captured pointers),
- the direction of data usage (input, output, and local data),
- canonical loop form according to the OpenMP standard,
- the ability to express properties of memory locations with DVMH directives,
- the ability to collapse iteration spaces of nested loops into one larger iteration space.

Analysis of the NAS Parallel Benchmarks 3.3.1

Automatic IR-level inline expansion:

- determine function calls that degrade analysis and schedule them to inline,
- do not affect the source code.

Extended analysis of privatizable, reduction and induction scalars which participate in address arithmetic:

- local IR-level transformation to break the explicit relation between scalars and MPI functions,

```
 double sum = 0.0E0;

 for (int j = 1; j <= lastcol - firstcol + 1; ++j)
   sum = sum + r[j - 1] * r[j - 1];

MPI_Send(&sum, 1, dp_type, reduce_exch_proc[i - 1],
         i, MPI_COMM_WORLD);
```

```
double sum = 0.0E0;
double sum_promoted = sum;
for (j = 1; j <= lastcol - firstcol + 1; ++j)
  sum_promoted = sum_promoted + r[j - 1] * r[j - 1];
sum = sum_promoted;
MPI_Send(&sum, 1, dp_type, reduce_exch_proc[i - 1],
         i, MPI_COMM_WORLD);
```

- LLVM-based analysis of memory references promoted to be register references.

Pre-specified data usage direction for the standard procedures: intrinsic Fortran procedures, functions from the C standard library and **MPI procedures**.

Dynamic analysis to reveal privatizable arrays and manual specification of analysis options to assume that subscript expression is in bounds value and to prevent math functions to indicate errors by setting *errno*.

Manual Transformation of the EP and BT Benchmarks

Replacement of a reduction array of the constant size with scalar variables (EP):

```
q[l] = q[l] + 1.0;
```

⇨

```
switch (l) {
  case 0: q0 = q0 + 1.0; break;
  case 1: q1 = q1 + 1.0; break;
   ...
}
```

Elimination of a large privatizable array to reduce memory usage on GPU (EP, BT):

```
for (i = 0; i < NK; i++) {
  ...
  x[i] = r46 * (*x4);
}

for (i = 0; i < NK; i++) {
  x1 = 2.0 * x[2 * i] - 1.0;
  x2 = 2.0 * x[2 * i + 1] - 1.0;
  ...
}
```

⇨

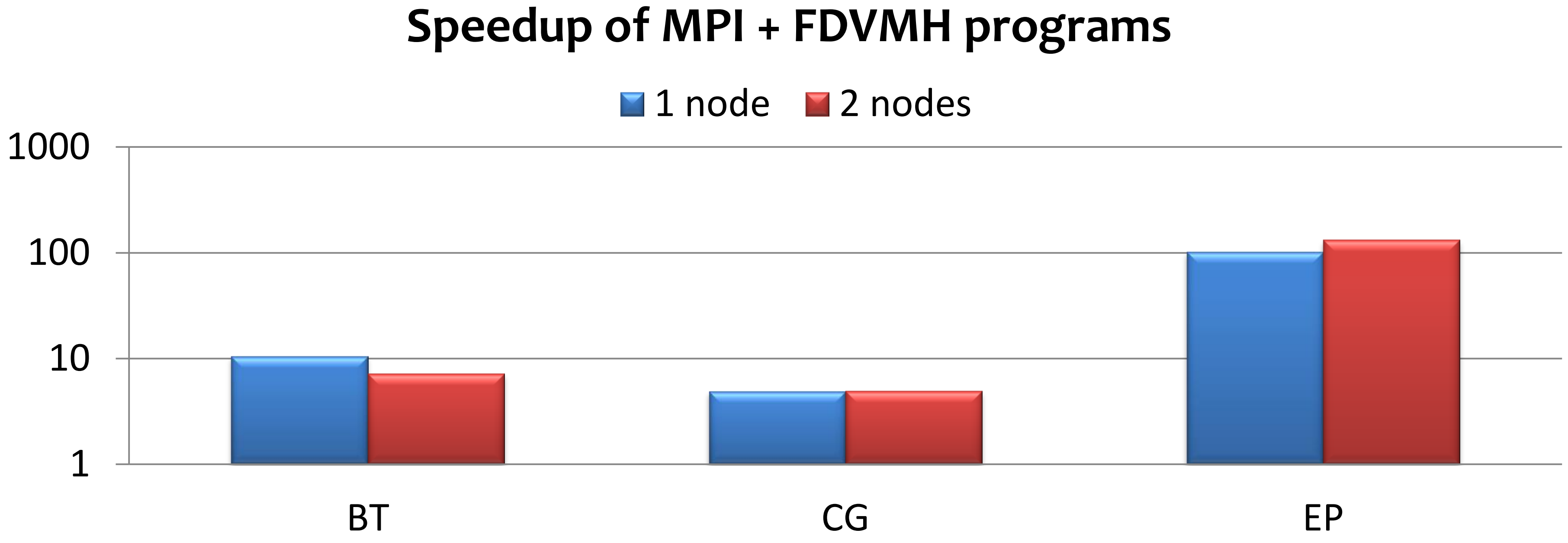
```
for (i = 0; i < NK; i++) {
  double x_2i, x_2i1;
  ...
  x_2i = r46 * (*x4);
  ...
  x_2i1 = r46 * (*x4);

  x1 = 2.0 * x_2i - 1.0;
  x2 = 2.0 * x_2i1 - 1.0;
  ...
}
```

Parallelization of the NAS Parallel Benchmarks 3.3.1 (Fortran)

Time (sec)

	BT	CG	EP	BT (MPI+DVMH)	CG (MPI+DVMH)	EP (MPI+DVMH)
1 node	665.1	397.5	93.6	63.3	80.99	0.62
2 nodes	361.6	209.6	46.5	50.3	42.6	0.38

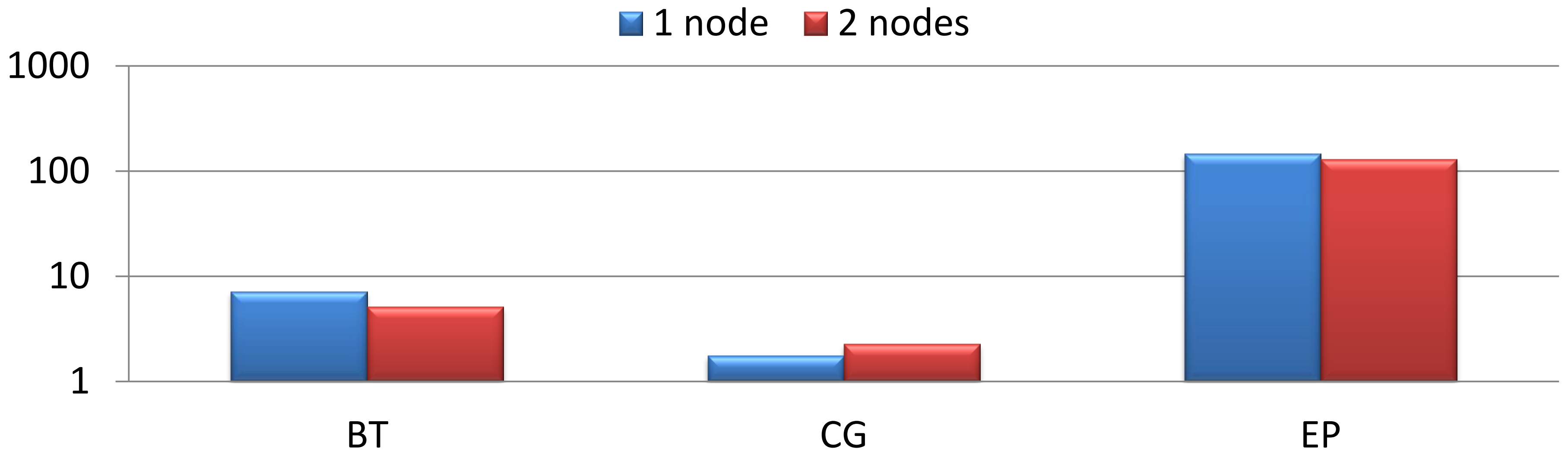


Parallelization of the NAS Parallel Benchmarks 3.3.1 (C)

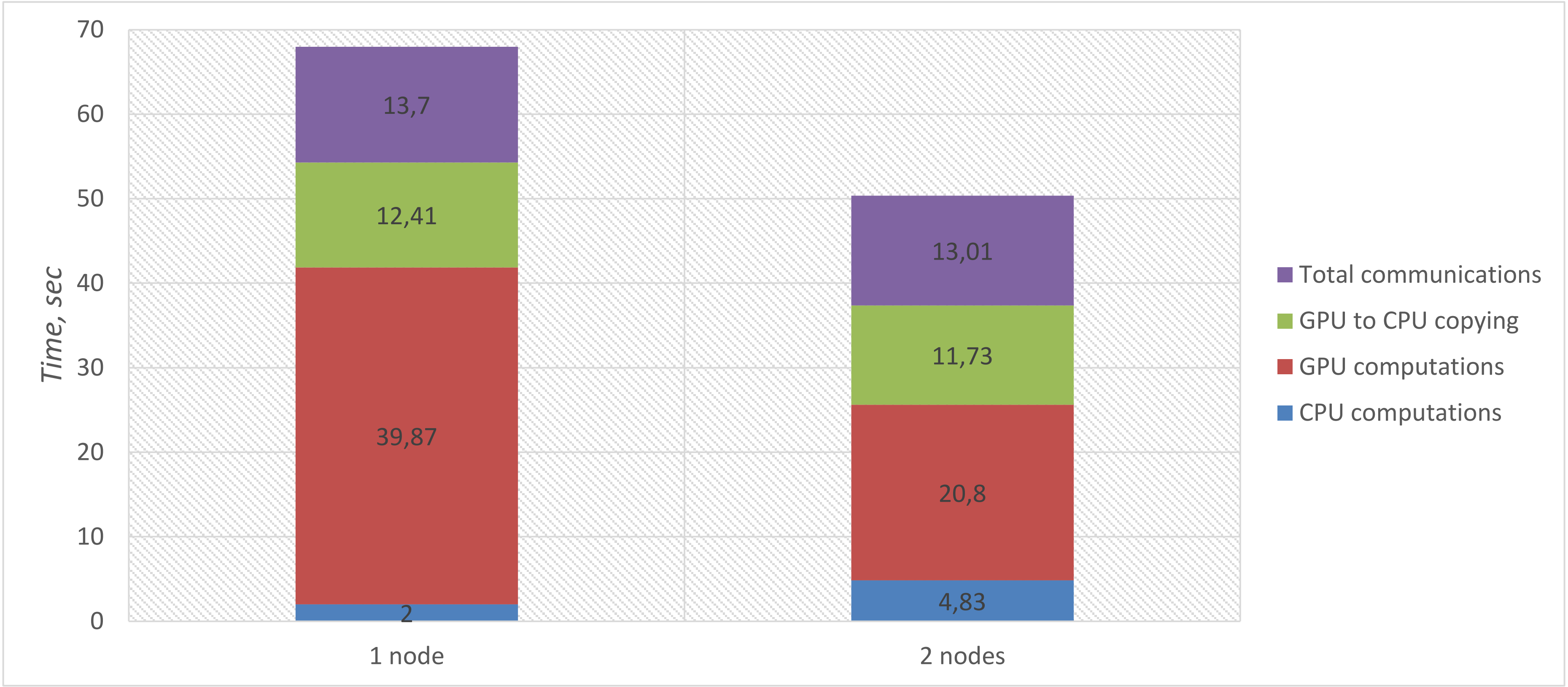
Time (sec)

	BT	CG	EP	BT (MPI+DVMH)	CG (MPI+DVMH)	EP (MPI+DVMH)
1 node	694.6	326.1	98.4	97.7	186.12	0.67
2 nodes	386	218.9	49.2	75.7	96.75	0.38

Speedup of MPI + CDVMH programs



The Ratio of Computation Time to Communication Time in the BT Benchmark (Fortran)



Conclusion

SAPFOR relies on the new features of the DVM system that allows us to offload computations in MPI programs to a GPU in a semi-automatic way.

The SAPFOR system implements an automatic parallelizing compiler that is suitable to parallelize well-formed programs.

The user may assert program properties or guide SAPFOR through the sequence of source-to-source transformations.

To gain parallel program performance the SAPFOR system can rely on various optimizations implemented in the DVMH compiler and runtime system, the DVM system provides performance analysis tools that operate in terms understandable to a user.

The SAPFOR and DVM systems can significantly reduce the effort required to embed intra-node parallelism into the existing MPI programs and to utilize available architectures such as multi-core CPUs or GPUs.

Thank you for your attention

http://dvm-system.org

https://github.com/dvm-system