A High-Level Approach for Parallelizing Legacy Applications for Multiple Target Platforms

Ritu Arora

Texas Advanced Computing Center

Email: rauta@tacc.utexas.edu

September 23, 2014

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

From a Report from the Council on Competitiveness (Compete.org)

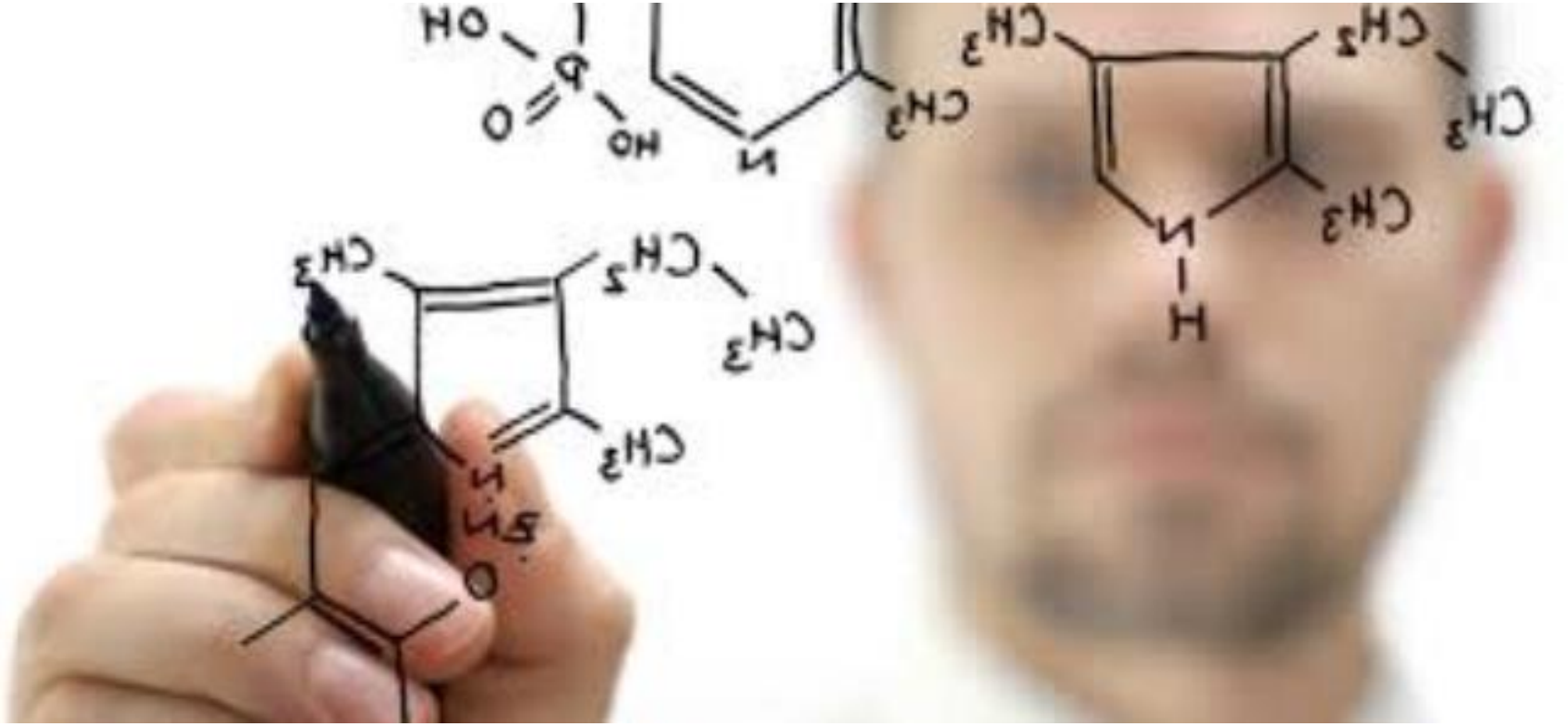
4) Three Systemic Barriers Are Stalling HPC Adoption: Lack of Application Software, Lack of Sufficient Talent, and Cost Constraints

"The problem is price and the need for a dedicated technical person."

"Lack of in-house expertise is a problem for us."

"It could be beneficial to our business if an appropriate return on investment could be realized for the acquisition [of] and training on these tools."

Domain Experts Lacking Access to HPC Experts



- What do they really want to focus on?
  - Get the science done quickly or spend time in learning low-level details of different parallel programming paradigms?
  - Leverage the investments made in legacy application development or invest in HPC application development from scratch?

Which are the most widely used parallel programming paradigms for High Performance Computing applications?

MPI

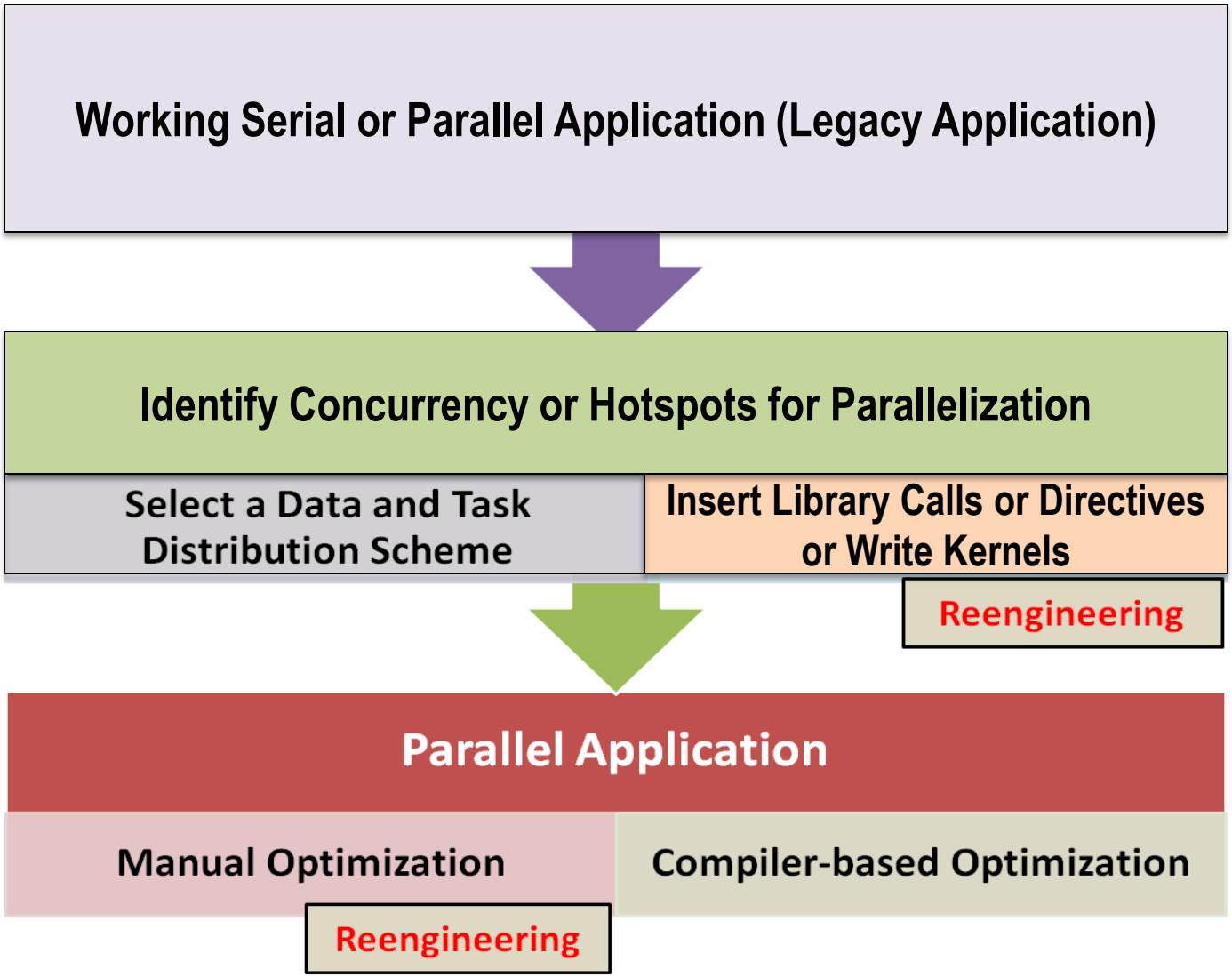
OpenMP

CUDA

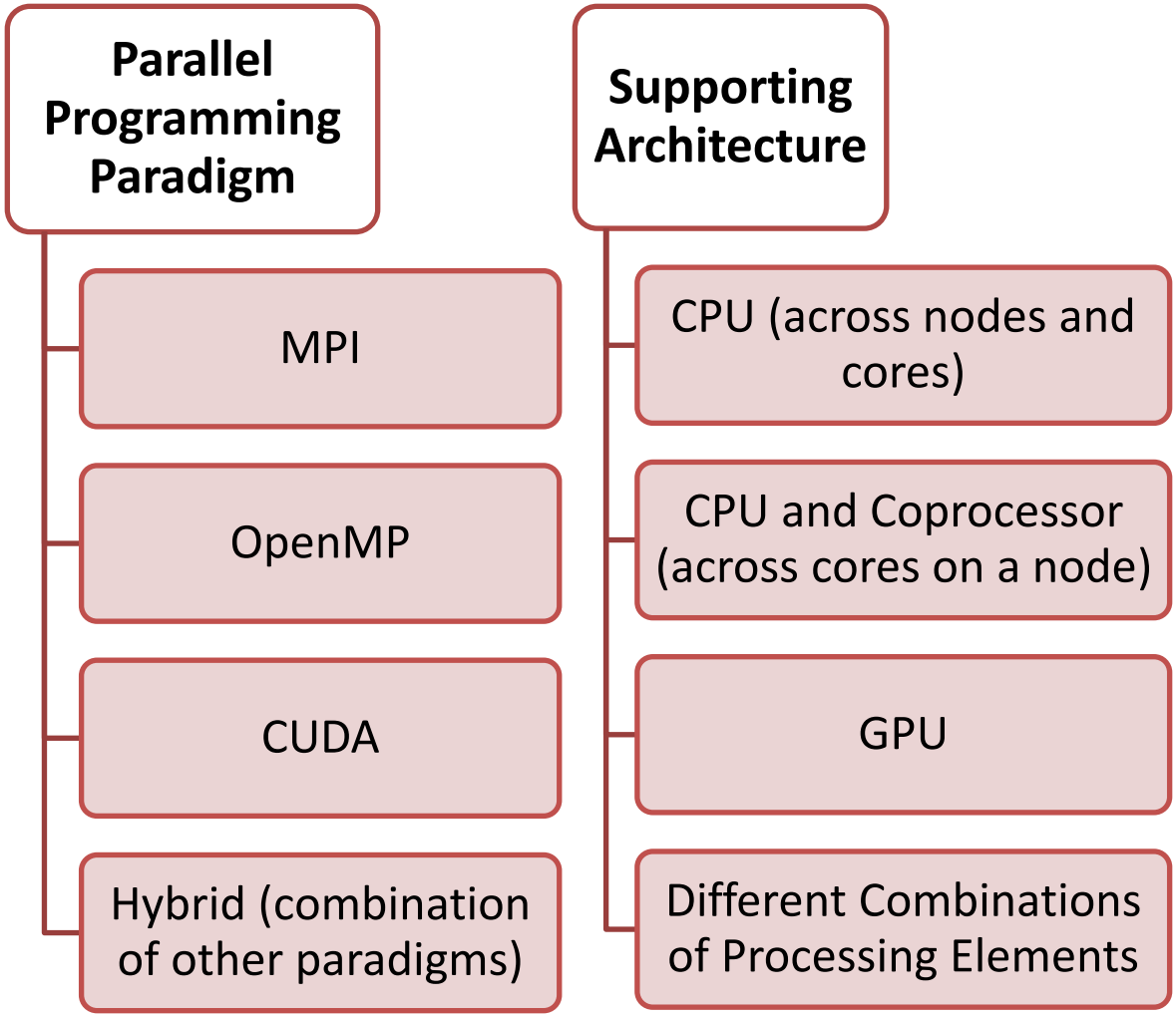
Hybrid programming

These paradigms can be classified under the category of **explicit parallelization.**

Traditional Process of Explicit Parallelization



Mastering Multiple Paradigms for Explicit Parallelization is Often a Challenge for Several Domain-Experts

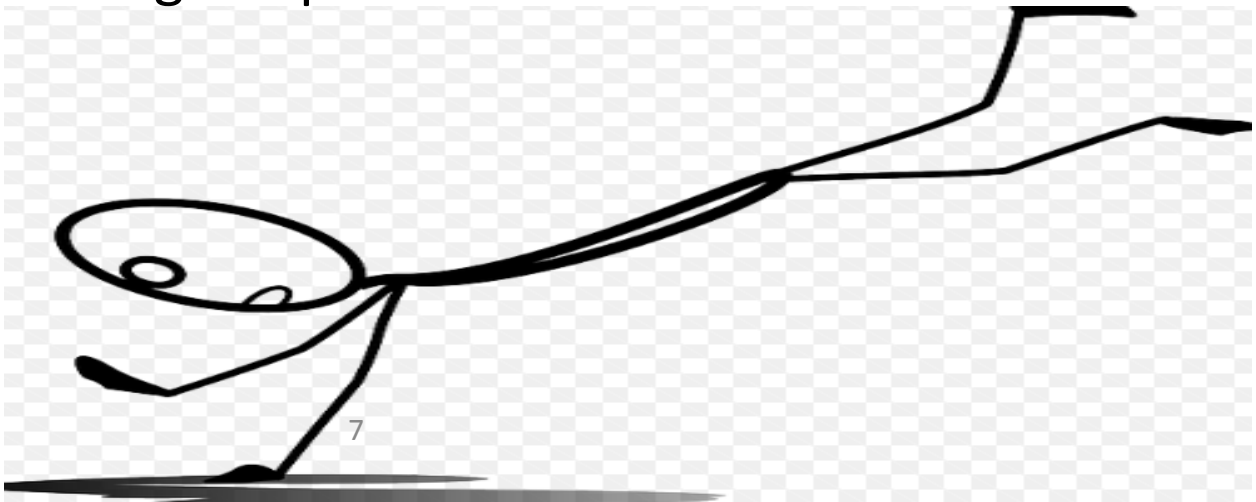


We are only talking about explicit parallelization here. What about **architecture-specific optimizations?**

Why is Traditional Explicit Parallelization Using Different Paradigms a Challenge?

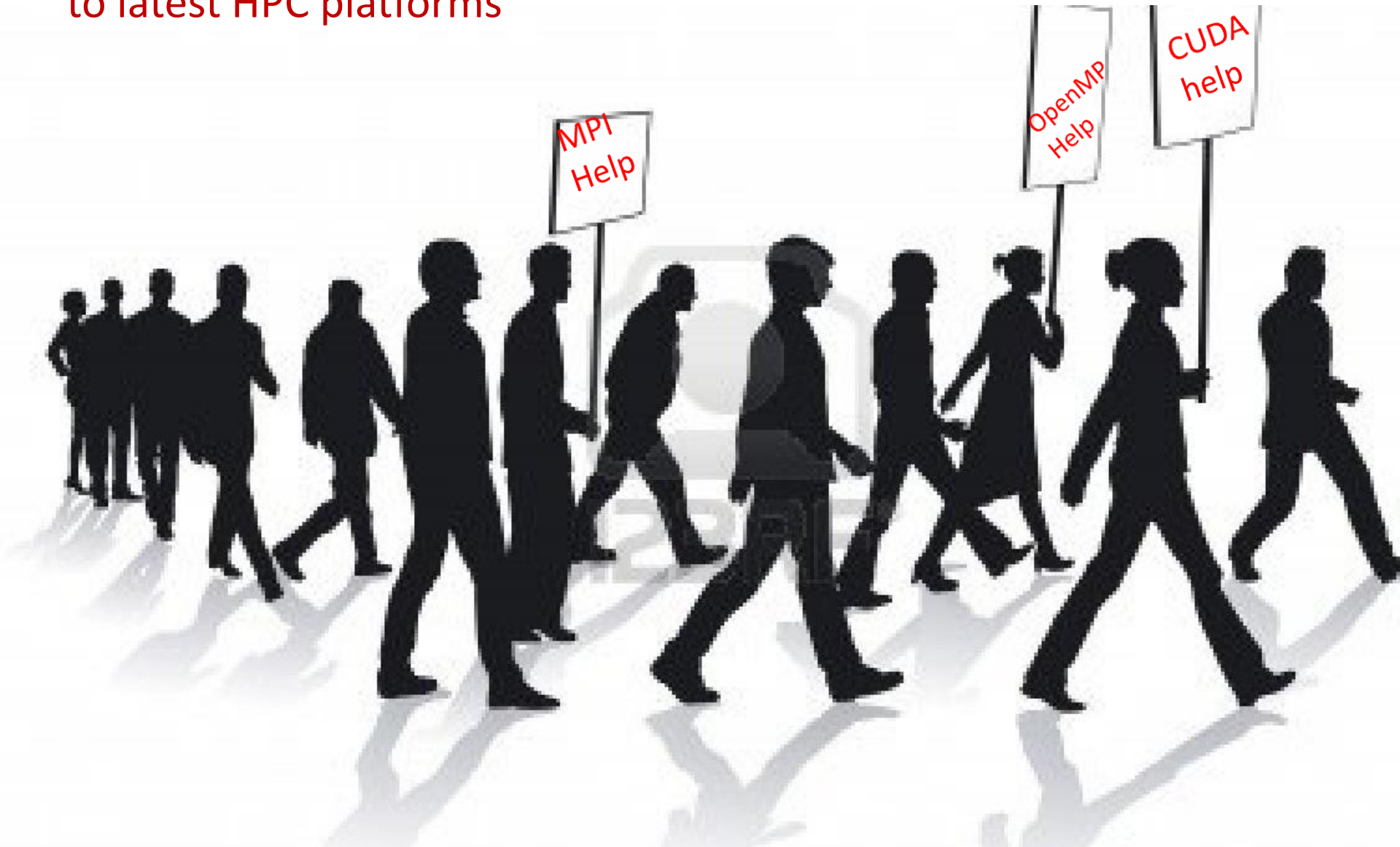
- Manual reengineering of legacy applications can be an effort- and time-intensive activity even for large and well-funded teams
  - First, need to understand the microarchitectural details of the latest HPC platforms
  - Then, learn about the details of the supported parallel programming paradigms
  - Then, invest time and effort in manually reengineering the code

- What if you do not see good performance at the end of the whole exercise?

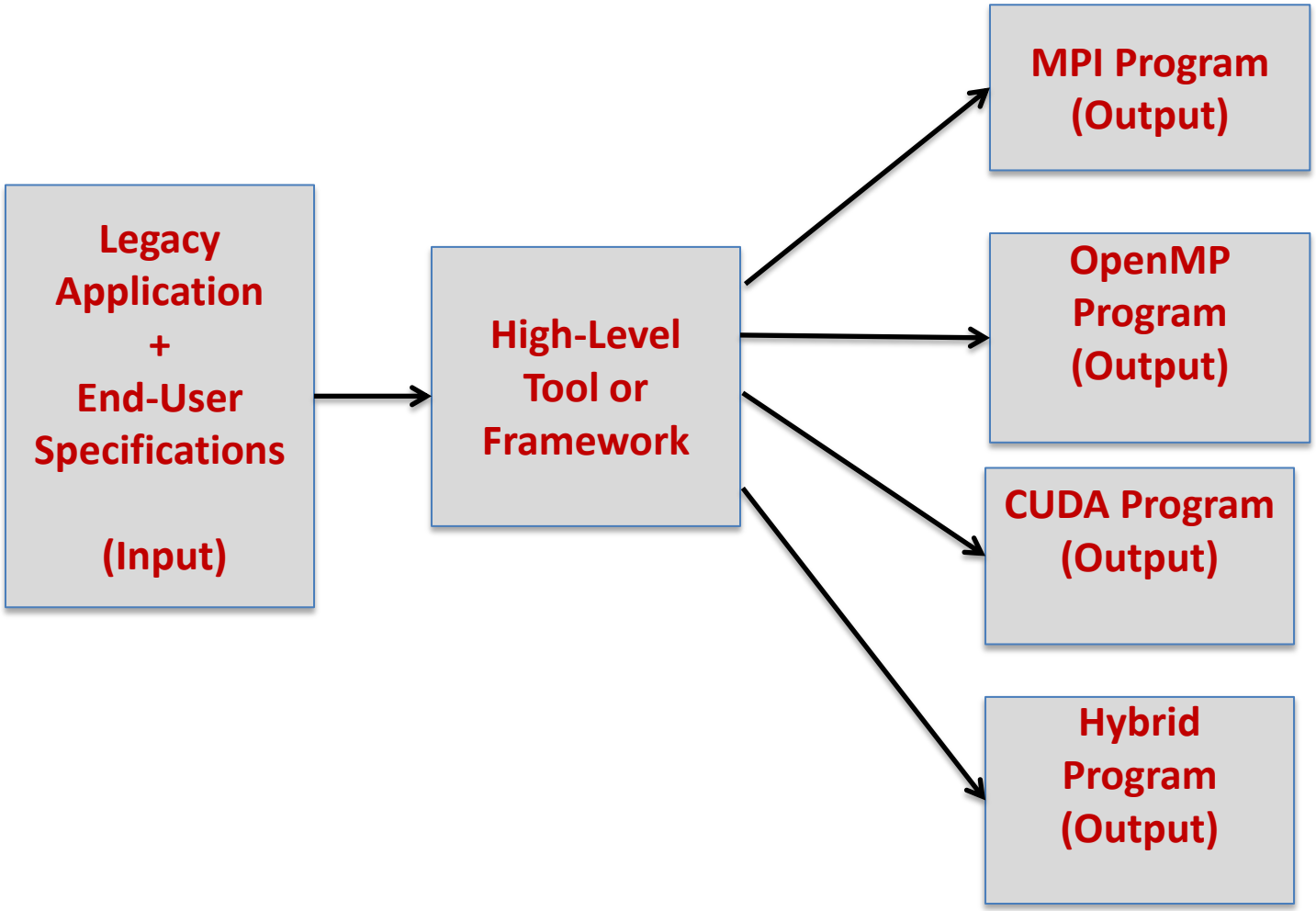


TACC

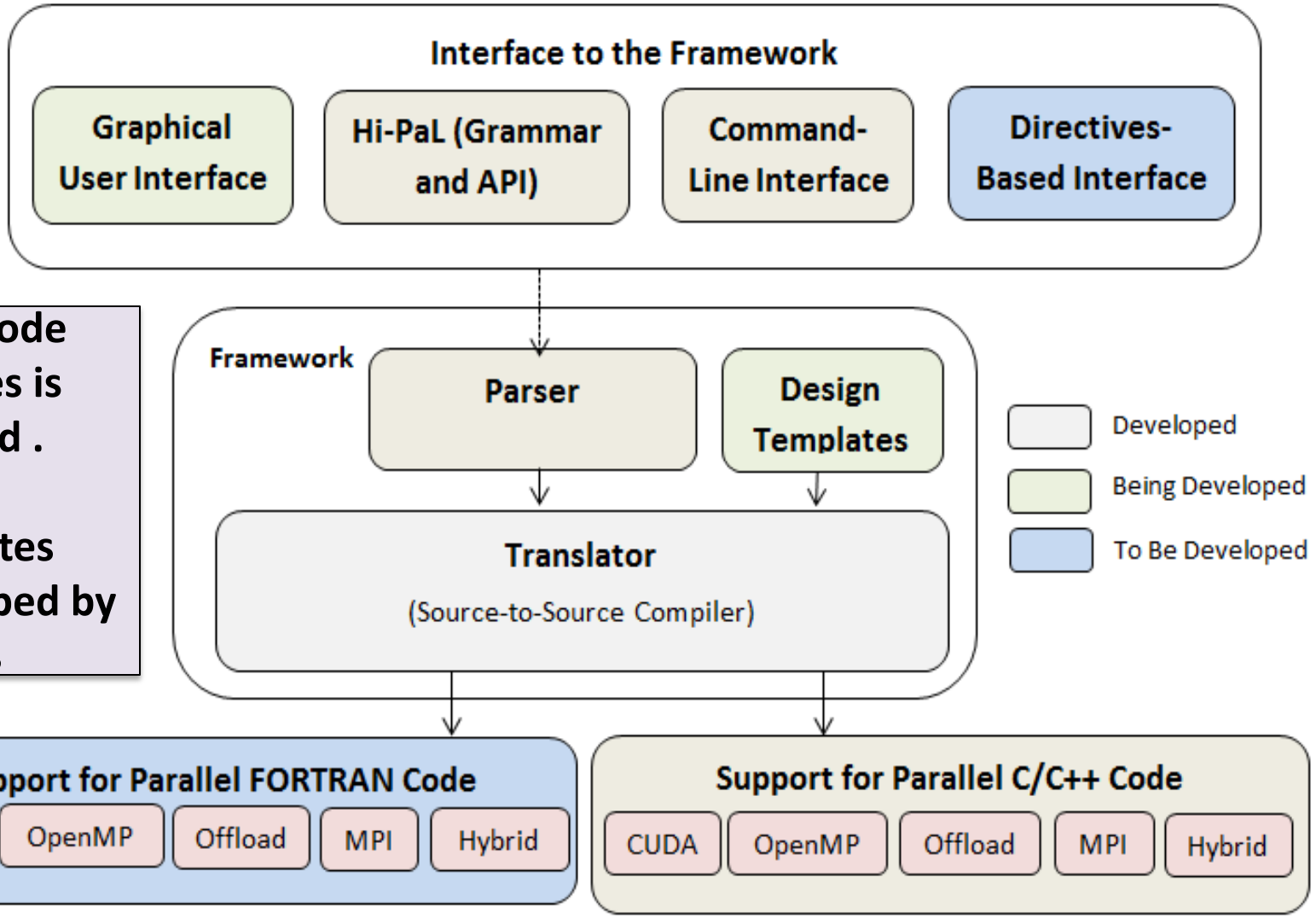
- There is a need for a high-level approach that can offer a low-risk way for domain-experts to try HPC
- There is a need for a tool that can help in porting legacy applications to latest HPC platforms



Vision for the Desired High-Level Solution for Explicit Parallelization



Explicit Parallelization at a High-Level



- **Static code analyses is involved .**
- **Design Templates developed by experts**

What is the secret sauce that went into the design and development of our high-level tool ?

1. Encapsulated the knowledge of expert parallel programmers inside design templates and rules that are used for source-to-source transformation
2. Abstracted the commonly seen standard and non-standard steps involved in explicit parallelization
3. Adopted the user-guided approach instead of 100% automation

Standard and Non-Standard Steps for Parallelization that are Repeatable

- Examples of standard steps in developing an MPI application (common in all MPI programs)
  - Every MPI program has **`#include "mpi.h"`**
  - Every MPI program has **`MPI_Init`** and **`MPI_Finalize`** function calls

- Non-standard steps in developing an MPI application
  - for-loop parallelization, data distribution, mapping of tasks to processes, and orchestration of exchange of messages
    - Steps for splitting the work in a for-loop amongst all the processes in `MPI_COMM_WORLD` are standard for a given load-balancing scheme

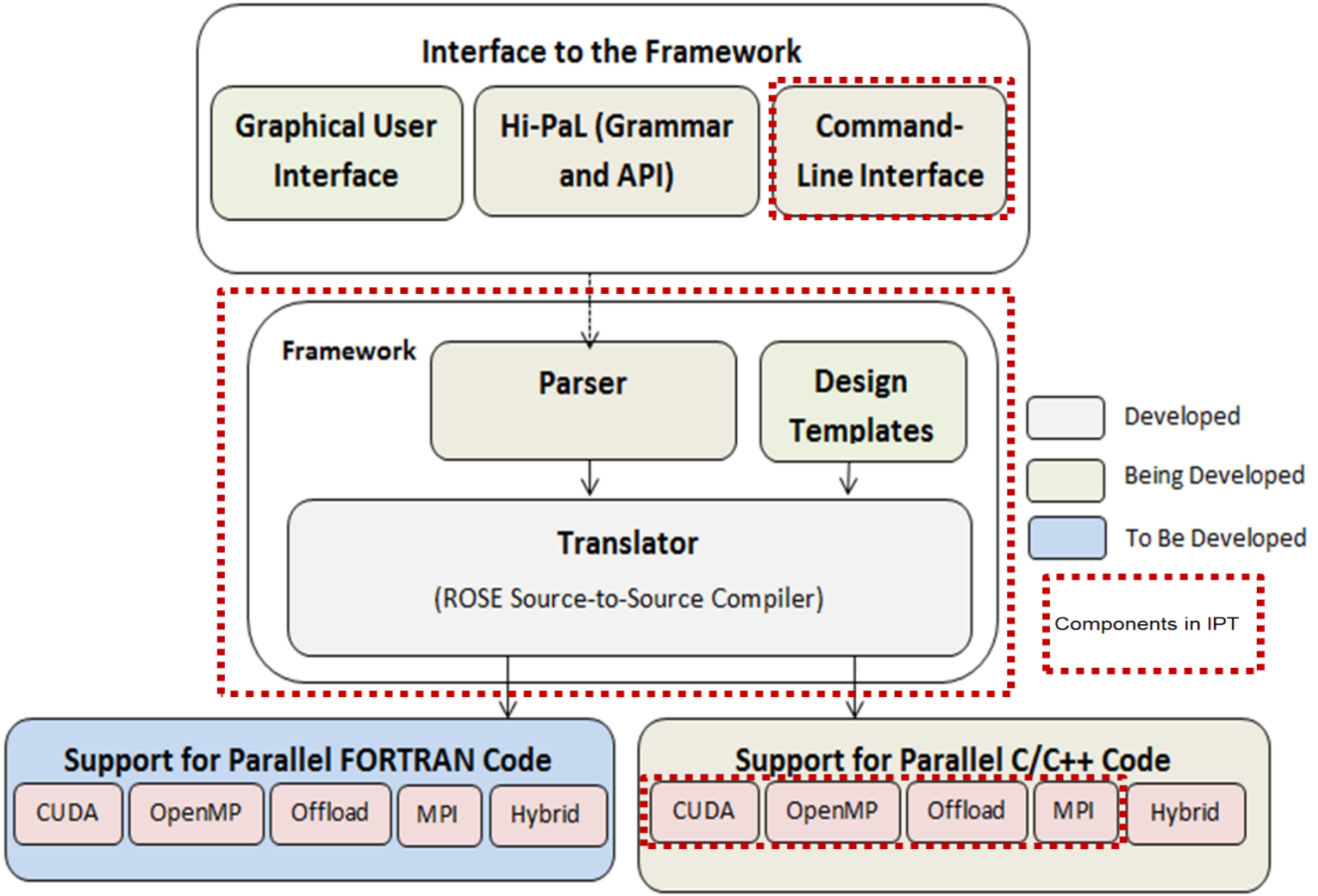
Which Interface is the right one?

- Command-Line Interface or the Interactive Parallelization Tool (IPT)
  - Most light-weight
  - Convenient for small applications and low barrier to adoption
- Graphical User Interface or GUI
  - Hides the transformation details from the end-user (those that they really do not want to know about) but is heavier than IPT for remote usage
  - Convenient for small applications and low barrier to adoption
- Domain-Specific Language or Hi-Pal interface
  - Convenient for large applications with repeated patterns and cross-cutting concerns but involves a small learning curve

Parallel Programming Via IPT

- Our Interactive Parallelization Tool (IPT) can be used for transforming legacy C/C++ programs into multiple parallel variants
  - Support for Fortran applications will be added in future
- IPT can be used to teach and learn different parallel programming paradigms through comparison and demonstration
  - without worrying about the low-level details related to the syntax and semantics of different paradigms
- IPT can help in porting legacy applications to latest architectures
- IPT shortens the application development cycle and hence can quickly show the impact of the design choices on performance
  - impact of static load balancing versus dynamic load-balancing
  - impact of choosing MPI only versus choosing hybrid programming

IPT Architecture



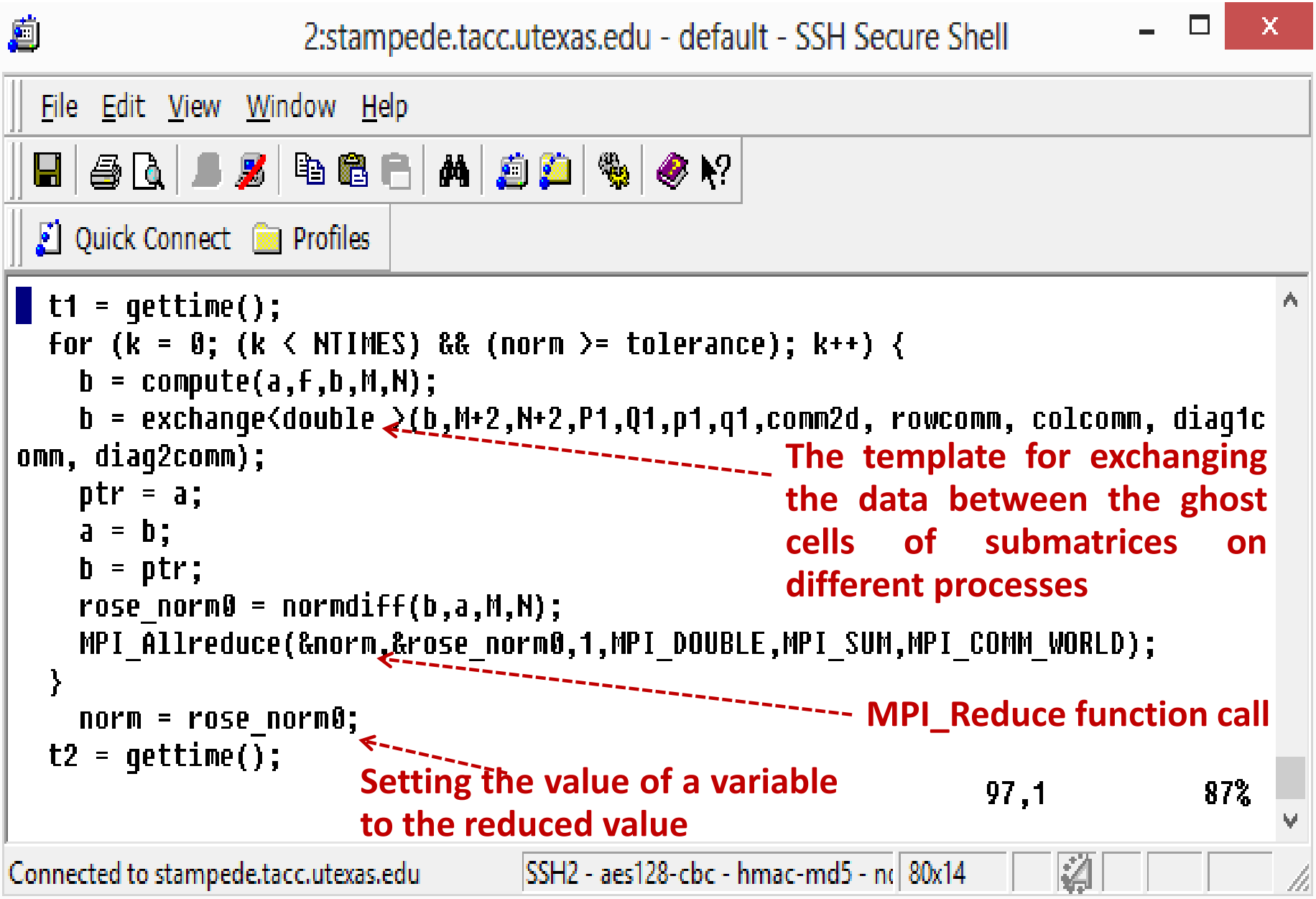
A Small Parallelization Exercise

```
1. //other code
2. NTIMES = atoi(argv[3]);
3. a = allocMatrix<double>(a, M, N);
4. b = allocMatrix<double>(b, M, N);
5. f = allocMatrix<double>(f, M, N);
6. start = 0;
7. //other code
8. printMatrix<double>(a, M, N);
9. t1 = gettime();
10. for (k = start; k < NTIMES && norm >= tolerance; k++) {
11.    b = compute(a, f, b, M, N);
12.    ptr = a;
13.    a = b;
14.    b = ptr;
15.    norm = normdiff(b, a, M, N);
16. }
17. t2 = gettime();//other code
```

Code snippet of serial Poisson Solver Code

Video Demo

Generated MPI Code for the Exercise



Exchange Template

2:stampede.tacc.utexas.edu - default - SSH Secure Sh

File Edit View Window Help

Quick Connect Profiles

```
template <typename T>
T** exchange(T** data, int nrows, int ncols, int P, int Q, int p, int q,
             MPI_Comm comm2d, MPI_Comm rowcomm, MPI_Comm colcomm) {

  MPI_Aint sizeoftype;
  MPI_Datatype datatype, temptype, vectype;
  int next, prev, down, up;
  MPI_Request req[8];
  MPI_Status status[8];

  T tempvar = (T)NULL;
  datatype = get_mpi_type(tempvar);

  // Create datatype for the recvtype
  MPI_Type_vector(nrows-2, 1, ncols, datatype, &
  MPI_Type_extent(datatype, &sizeoftype) ;
  int blens[2] = {1, 1};
  MPI_Aint displ[2] = {0, sizeoftype};
  MPI_Datatype types[2] = {temptype, MPI_UB};
  MPI_Type_struct (2, blens, displ, types, &vect
  MPI_Type_commit(&vectype);

  MPI_Cart_shift(rowcomm, 0, -1, &prev, &next);
  MPI_Cart_shift(colcomm, 0, -1, &down, &up);
  // printf("[%d,%d]: next=%d prev=%d down=%d up=%d\n", p, q, next, prev, down, up);

  // send and receive the boundary rows
  MPI_Irecv(&data[0][1], ncols-2, datatype, up, 0, colcomm, &req[0]);
  MPI_Irecv(&data[nrows-1][1], ncols-2, datatype, down, 0, colcomm, &req[1]);

  MPI_Isend(&data[1][1], ncols-2, datatype, up, 0, colcomm, &req[2]);
  MPI_Isend(&data[nrows-2][1], ncols-2, datatype, down, 0, colcomm, &req[3]);

  MPI_Irecv(&data[1][0], 1, vectype, next, 0, rowcomm, &req[4]);
  MPI_Irecv(&data[1][ncols-1], 1, vectype, prev, 0, rowcomm, &req[5]);

  MPI_Isend(&data[1][1], 1, vectype, next, 0, rowcomm, &req[6]);
```

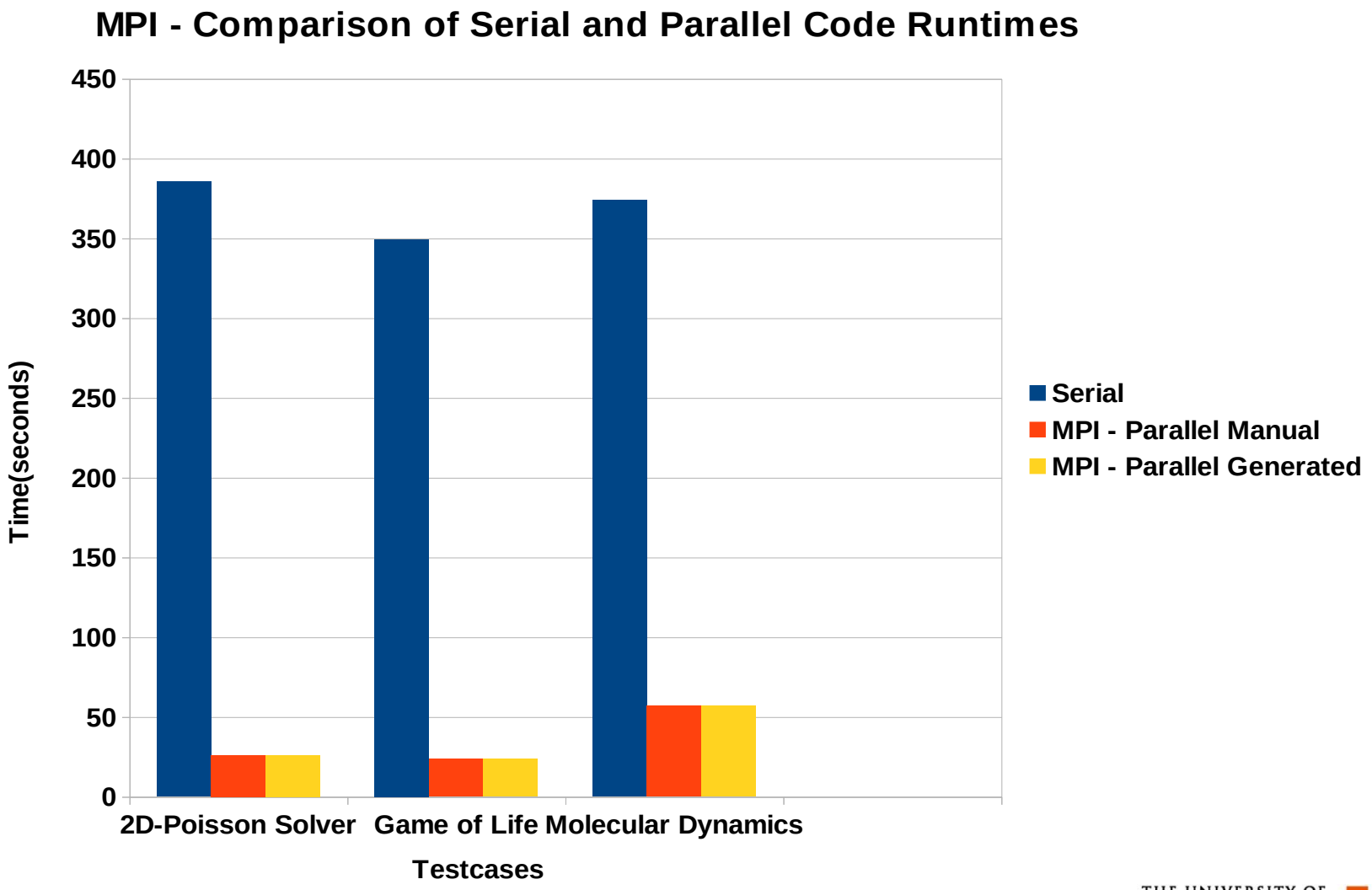
The generated code will have the call to the exchange template inserted. The exchange template has the code for exchanging the data amongst the ghost cells in a stencil-based code. The generated code is readable and is easy to understand as comments are inserted wherever necessary.

Connected to stampede.tacc.utexas.edu

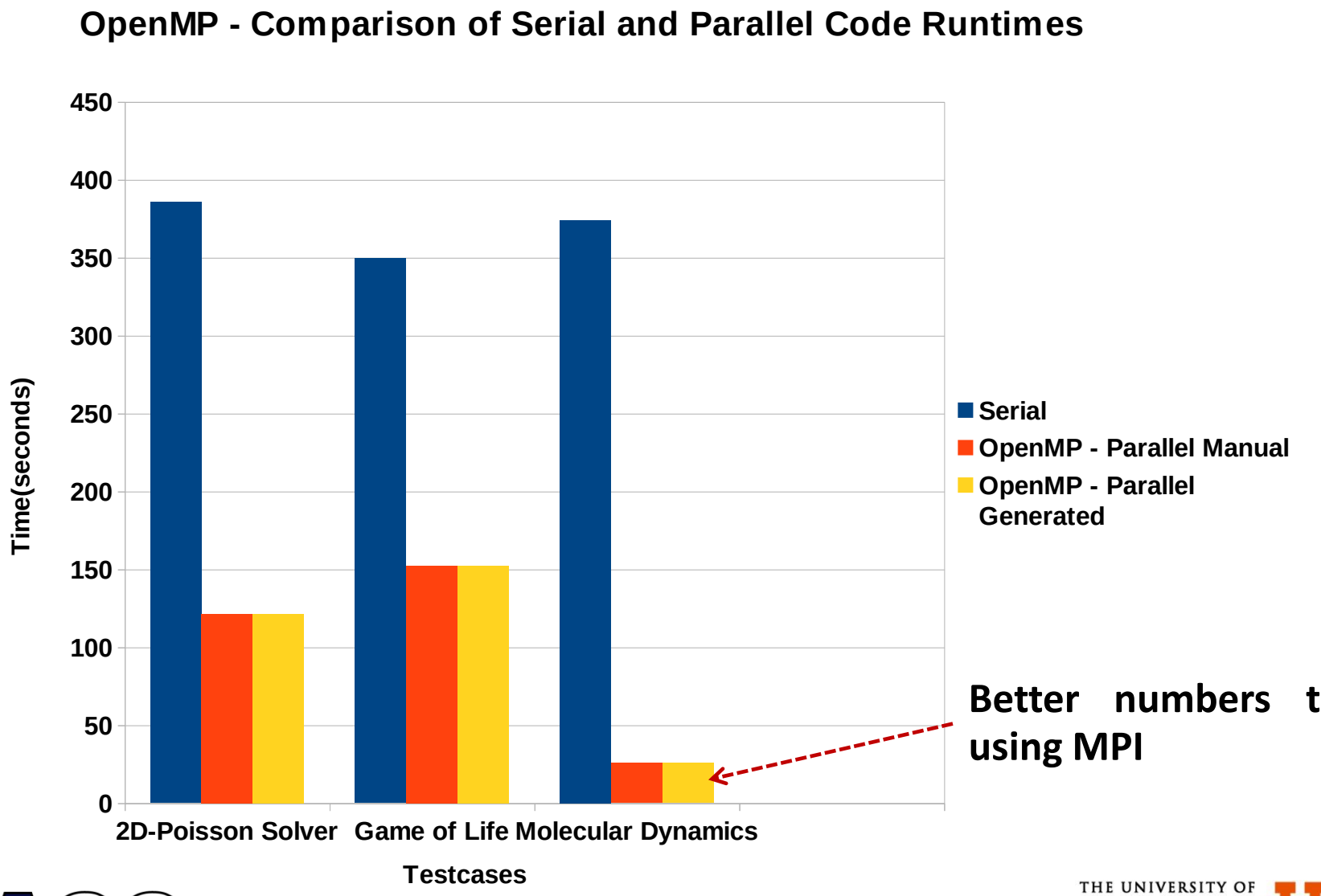
Benefits of IPT

- In how much time can you manually parallelize the Poisson Solver program using MPI?
  - IPT can help you in parallelizing this code in approximately 5 minutes given that you know the high-level concepts related to parallel programming, and are already familiar with IPT
  - IPT inserted about 357 lines of code in the serial version of the code in order to develop an MPI version
- In how much time can you learn a new parallel programming paradigm and use the knowledge gained in porting legacy application to a new platform manually?
  - IPT significantly reduces the time-to-solution
- The usability study to quantify the benefits related to IPT is pending

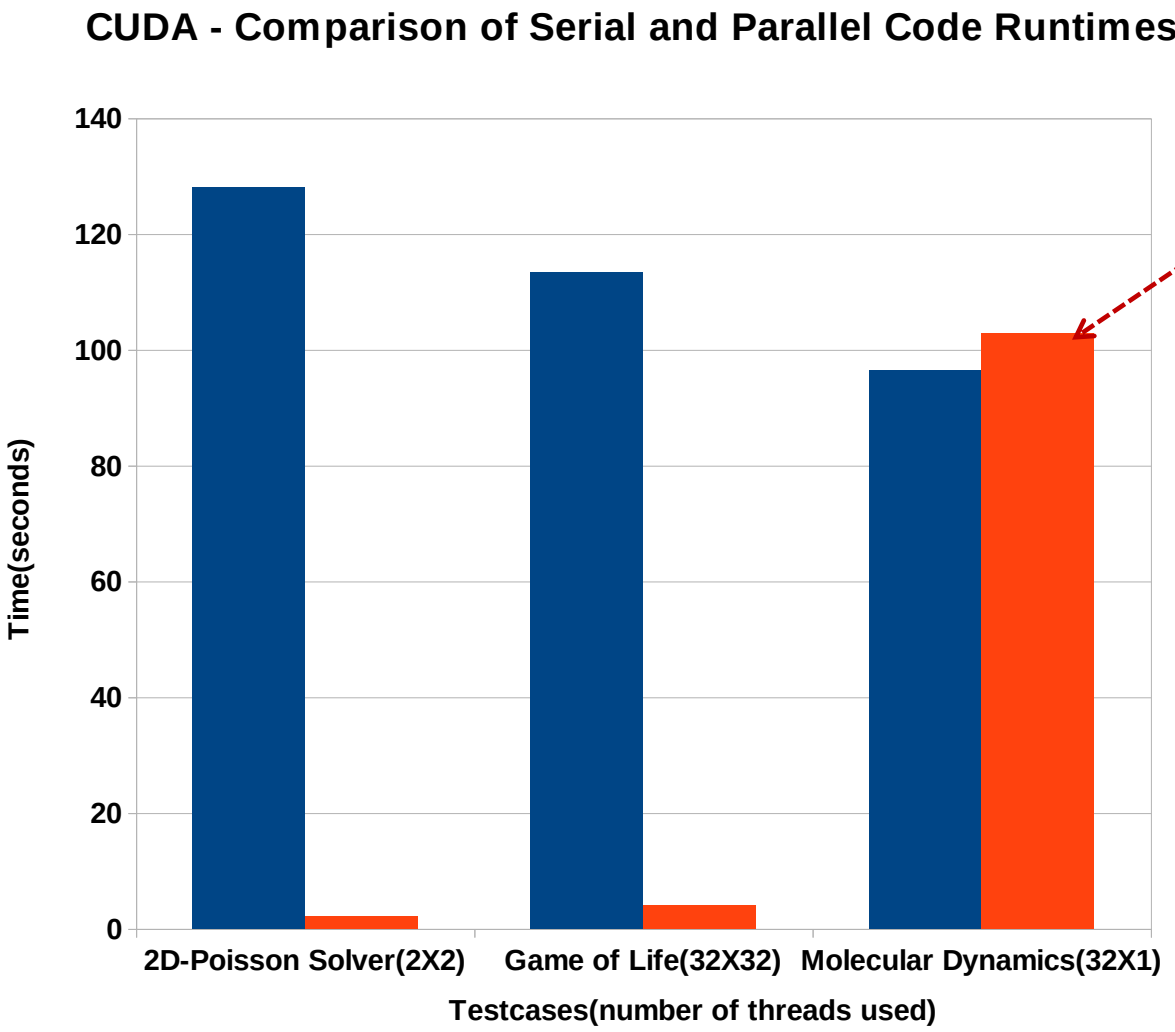
Results- Stencil-Based Pattern-16 MPI Processes



Results- Stencil-Based Pattern-16 OMP Threads



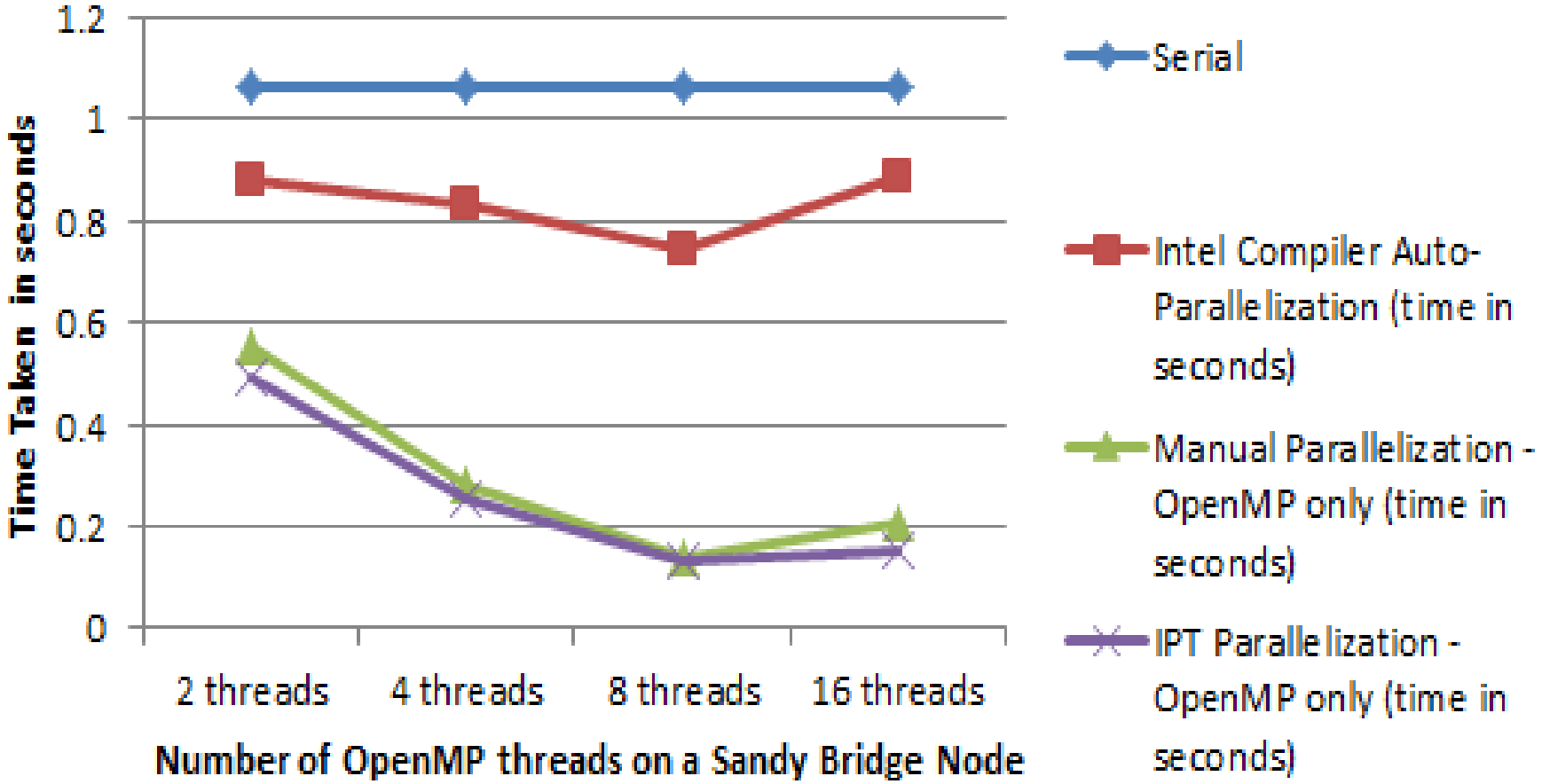
Results – Stencil-Based Computations - CUDA



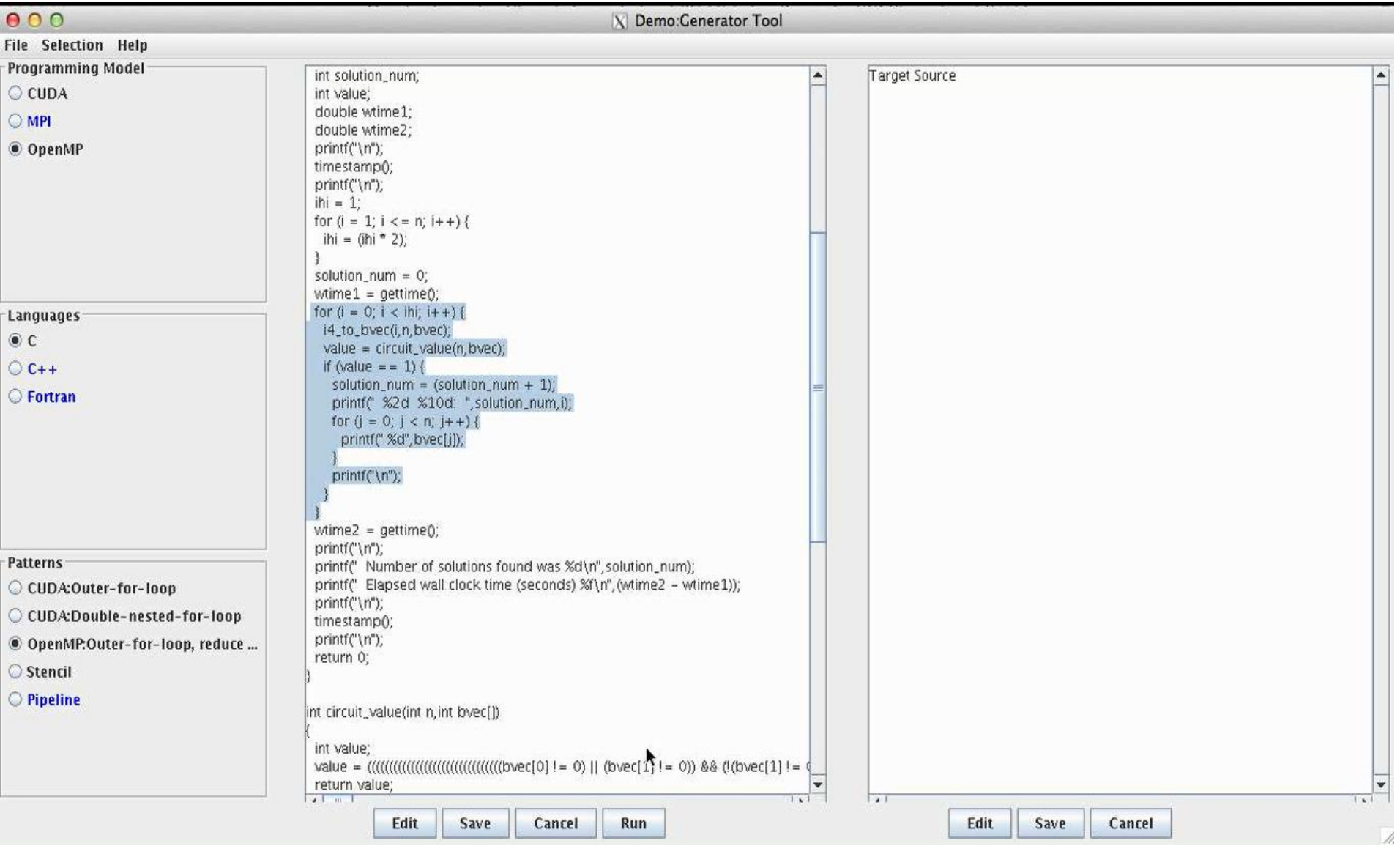
Due to deliberate choice of parallelizing a function that is better run on host

Slightly different versions of the code were used for CUDA code generation than for MPI and OpenMP code generation, problem size is different too

Run-time Comparison - Circuit Satisfiability Problem



Graphical User Interface (Demo)



Providing Specifications Through Hi-PaL (1)

```
Parallel section begins <hook type> (<hook pattern>)
mapping is <mapping type> {
 <Hi-PaL API for specifying the operation> <hook> &&
 in function (<function name>)
}
```

General Structure of Hi-PaL Code to Generate MPI Code

```
OMP_Parallel {
 <Hi-PaL API for specifying the operation> && schedule is
<schedule type> <hook> && in function (<function name>)
}
```

General Structure of Hi-PaL Code to Generate OpenMP Code

Providing Specifications Through Hi-PaL (2)

A set of Hi-PaL API has been developed for precisely capturing the end-users' specifications at a high-level

Hi-PaL API	Description
`ParExchange2DArrayInt(<array name>, <num of rows>, <num of columns>)`	Exchange neighboring values in stencil-based computations
`Parallelize_For_Loop where (<for_init_stmt>; <condition>; <stride>)`	Parallelize for-loop with matching condition, stride and initialization statement
`ReduceSumInt(<variable name>)`	`MPI_Reduce` with MPI_SUM operation or OpenMP reduction clause with '+' operator; reduced variable is of type integer

Parallelizing Poisson Solver (1)

```
1. //other code
2. NTIMES = atoi(argv[3]);
3. a = allocMatrix<double>(a, M, N);
4. b = allocMatrix<double>(b, M, N);
5. f = allocMatrix<double>(f, M, N);
6. start = 0;
7. //other code
8. printMatrix<double>(a, M, N);
9. t1 = gettime();
10. for (k = start; k < NTIMES && norm >= tolerance; k++) {
11. b = compute(a, f, b, M, N);
12. ptr = a;
13. a = b;
14. b = ptr;
15. norm = normdiff(b, a, M, N);
16. }
17. t2 = gettime();//other code
```

Code snippet of serial Poisson Solver Code

Parallelizing Poisson Solver (2)

```
1. Parallel section begins after ("NTIMES = atoi(argv[3]);")
mapping is Linear{
2.   ParExchange2DArrayDouble (a, M, N) before statement
     ("printMatrix<double>(a, M, N);")
     && in function ("main");
3.   ParExchange2DArrayDouble (b, M, N) before statement
     ("printMatrix<double>(a, M, N);")
     && in function ("main");
4.   ParExchange2DArrayDouble (b, M, N) after statement
     ("b=compute(a, f, b, M, N);") && in function ("main");
5.   AllReduceSumInt(norm) after statement
     ("norm = normdiff(b, a, M, N);") && in function ("main")
6. }
```

Hi-PaL Code to Generate MPI Code for Poisson Solver

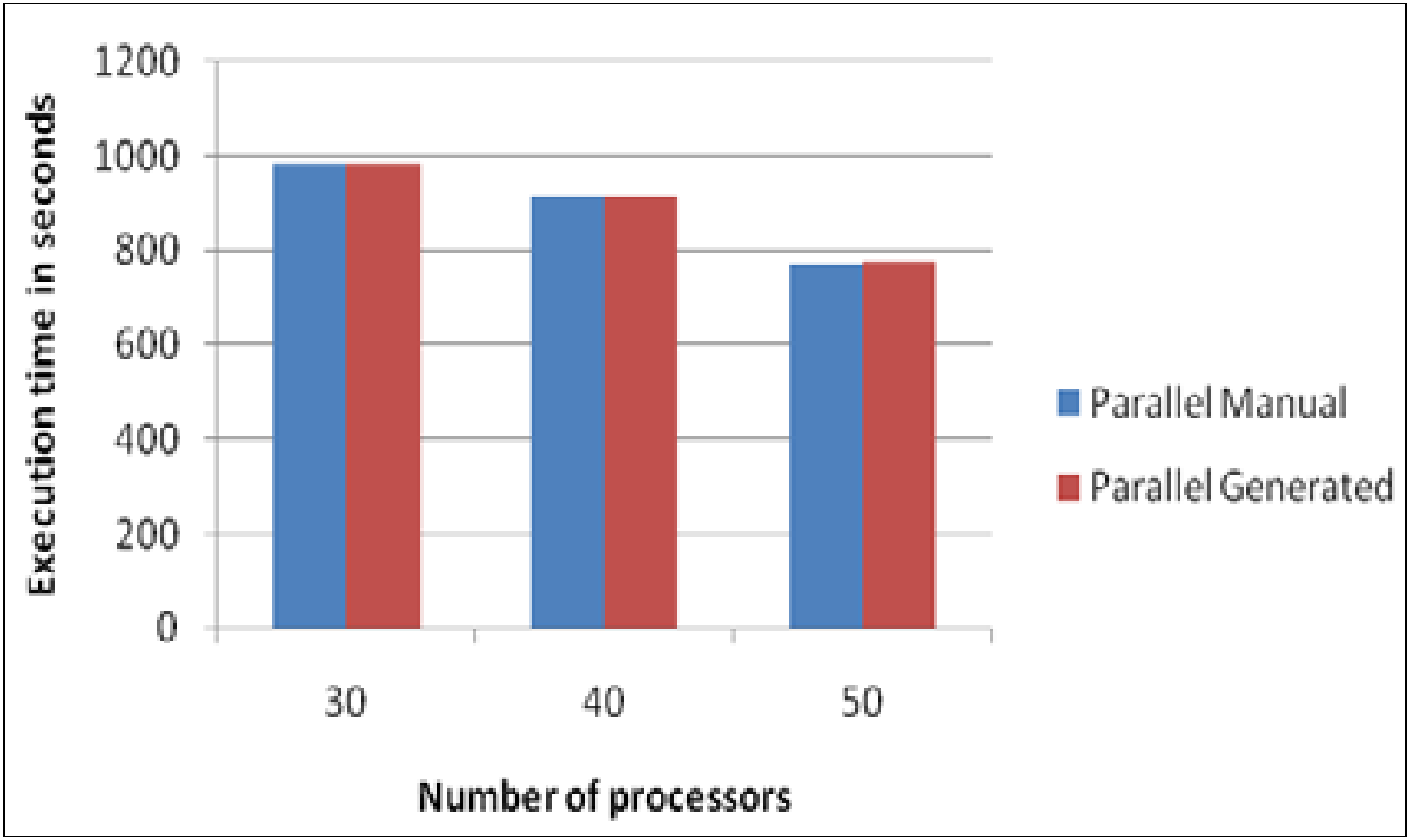
Generated MPI Code for Poisson Solver (1)

```
1. //other code
2. NTIMES = atoi(argv[3]);
3. MPI_Init(NULL, NULL);
4. MPI_Comm_size(MPI_COMM_WORLD, &size_Fraspa);
5. MPI_Comm_rank(MPI_COMM_WORLD, &rank_Fraspa);
6. create_2dgrid(MPI_COMM_WORLD, &comm2d_Fraspa,…);
7. create_diagcomm(MPI_COMM_WORLD, size_Fraspa, …);
8. rowmap_Fraspa.init(M, P_Fraspa, p_Fraspa);
9. colmap_Fraspa.init(N, Q_Fraspa, q_Fraspa);
10. myrows_Fraspa = rowmap_Fraspa.getMyCount();
11. mycols_Fraspa = colmap_Fraspa.getMyCount();
12. M_Fraspa = M;
13. N_Fraspa = N;
14. M = myrows_Fraspa;
15. N = mycols_Fraspa;
16. a = allocMatrix<double>(a, M, N);
```

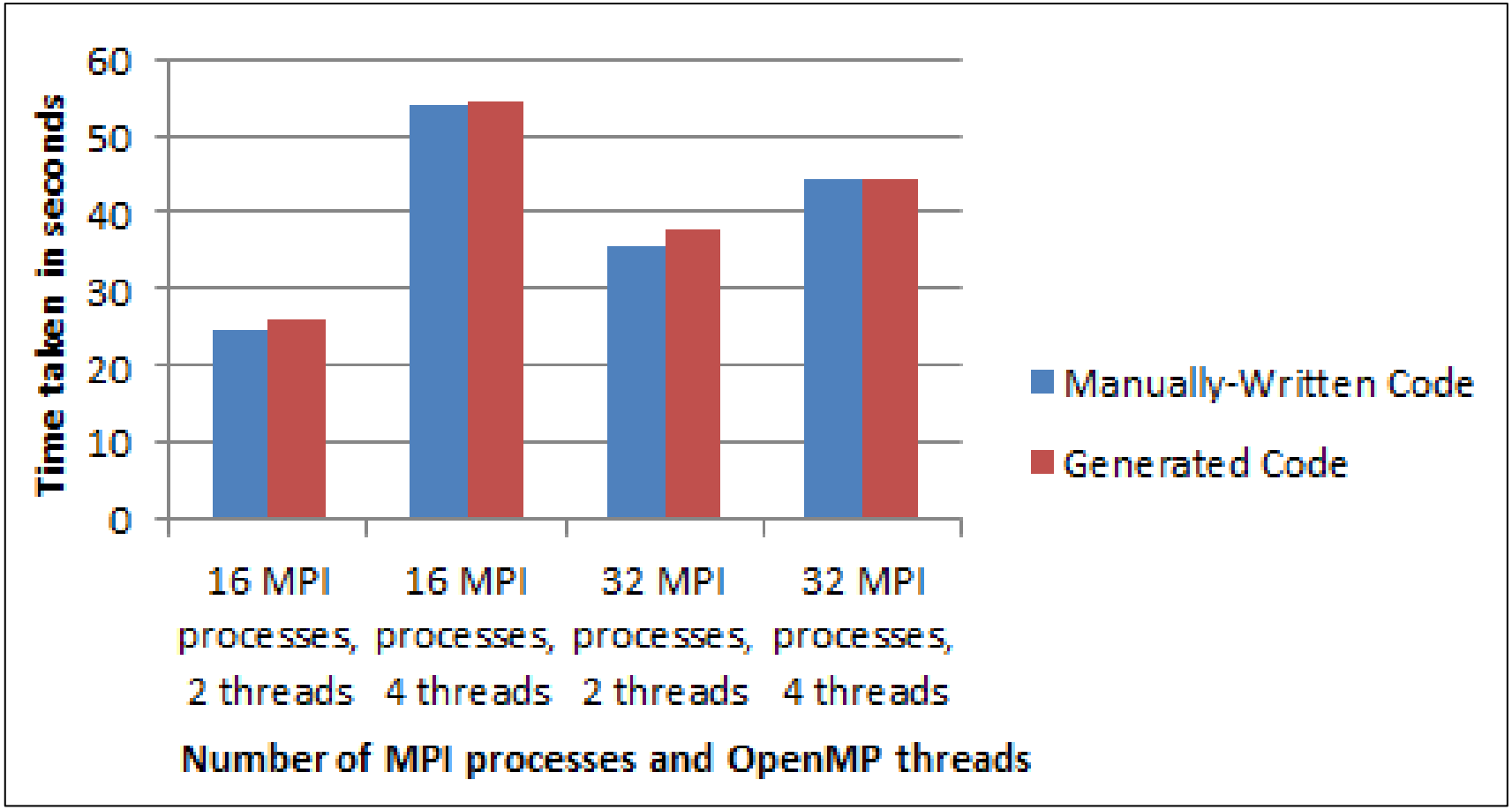
Generated MPI Code for Poisson Solver (2)

```
17. b = allocMatrix<double>(b, M, N);
18. f = allocMatrix<double>(f, M, N);
19. start = 0;
20. //other code
21. a = exchange<double>(a, myrows_Fraspa + 2, …);
22. b = exchange<double>(b, myrows_Fraspa + 2, …);
23. printMatrix<double>(a, M, N);
24. t1 = MPI_Wtime();
25. for (k = start; k < NTIMES && norm >= tolerance; k++) {
26. b = compute(a, f, b, M, N);
27. b = exchange<double>(b, myrows_Fraspa + 2, …);
28. ptr = a;
29. a = b;
30. b = ptr;
31. norm = normdiff(b, a, M, N);
32. MPI_Allreduce(&norm, &norm_Fraspa, 1, MPI_INT, MPI_SUM,…);
33. norm = norm_Fraspa;
34. }
36. //other code
```

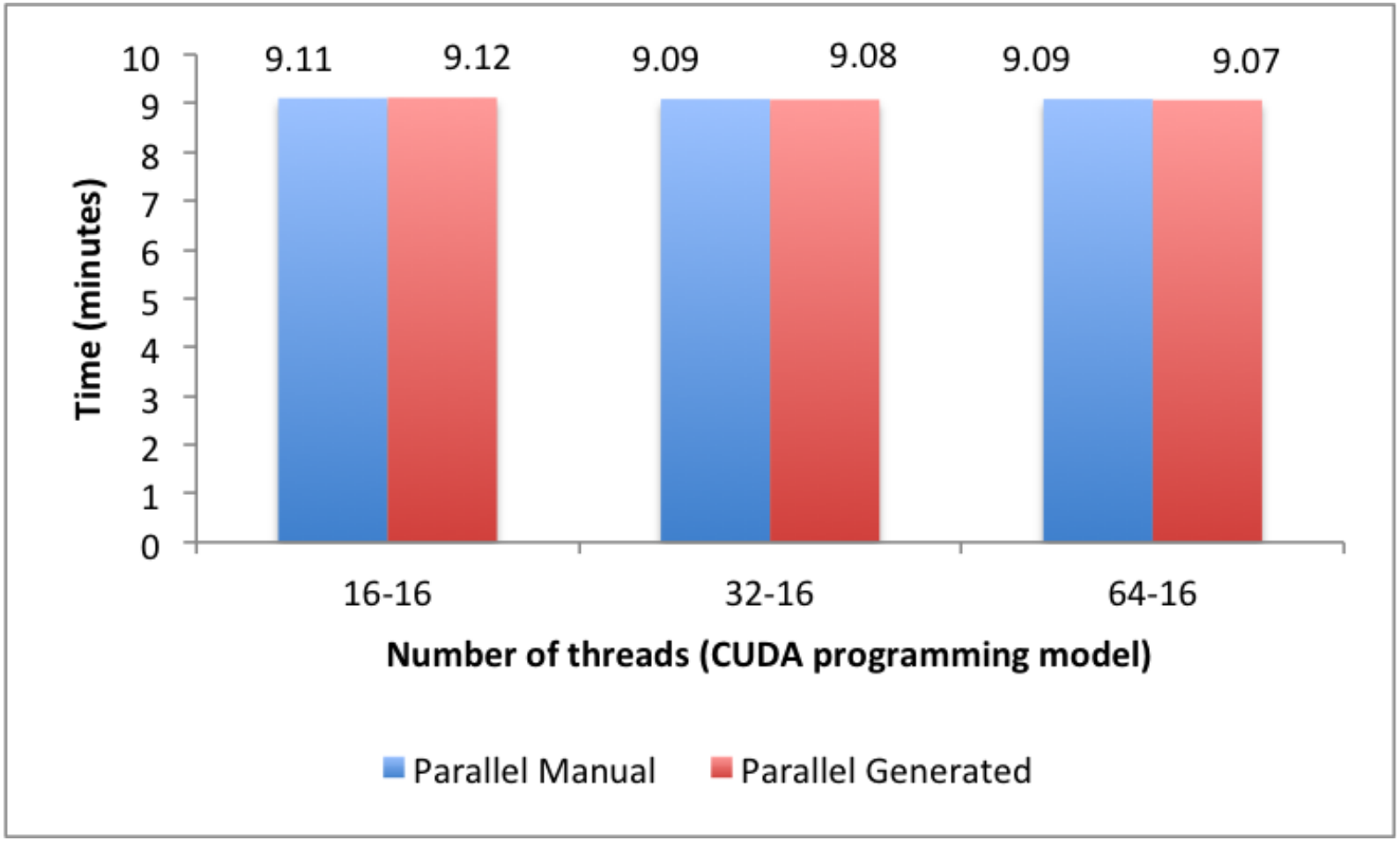
Results: Poisson Solver (**Hi-PaL based MPI**)



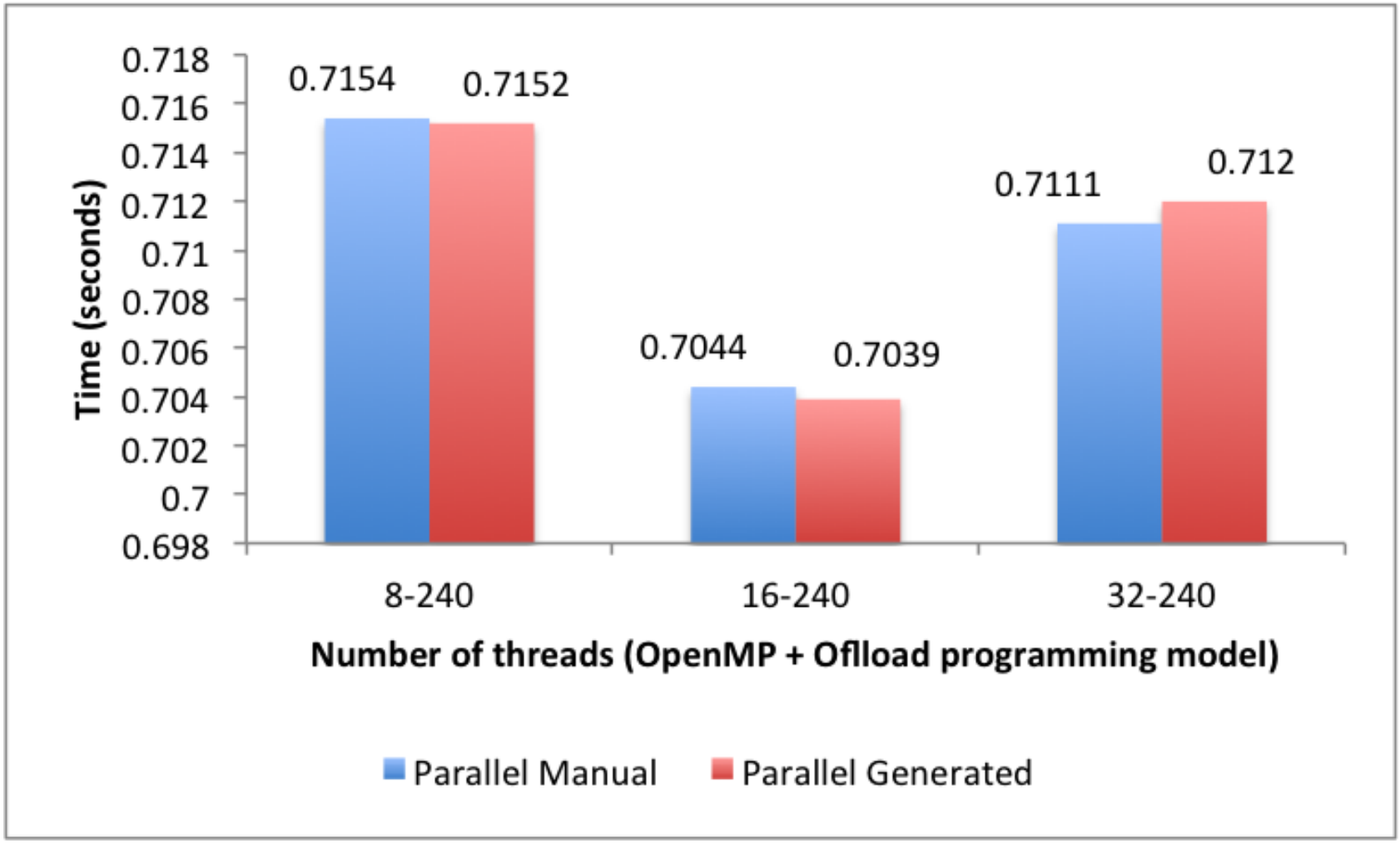
Results: Genetic Algorithm for Content Based Image Retrieval (**Hi-PaL based MPI & OpenMP**)



Results: Seismic Tomography Code (**GUI-based CUDA**)



Results: Circuit Satisfiability Code (**GUI-based OpenMP + Offload**)



Summary of Features & Benefits

- Enhances the productivity of the end-users in terms of the reduction in the time and effort
  - reduction in manual effort by over 90% while ensuring that the performance of the generated parallel code is within 5% of the sample hand-written parallel code
- Leverages the knowledge of expert parallel programmers
- Separates the sequential and parallel programming concerns while preserving the existing version of sequential applications

Ongoing and Future Work

- Code cleanup and preparation for the public release
- Support for handling irregular meshes will be added in future
- Support for Fortran code generation will be added in future as well
- Support for generating hybrid applications is available with Hi-PaL interface. The command-line interface and GUI will be extended to support hybrid code generation as well

Thanks!

Special thanks to:

Dr. Purushotham Bangalore, UAB

XSEDE for resources and interns

TACC's STAR Partners for supporting Interns

Sponsors of this workshop

Questions or Comments?

Separation of Sequential and Parallel Concerns