

PARSIT - A Parallel Algorithm Reconfiguration Simulation Tool

Gopal Racherla†*, Steven E. Killian†, Leslie D. Fife†, Michael A. Lehmann†, and Rajesh Parekh‡

†School of Computer Science
University of Oklahoma
200 Felgar St. Rm. 114
Norman OK, 73019 USA

‡Department of Computer Science
Iowa State University
104 Atanasoff Hall
Ames IA, 50011 USA

Abstract

We present a tool - PARSIT - to perform simulation and performance evaluation on dynamically reconfigurable architectures. PARSIT assumes a reconfigurable system with standalone processors connected using a multistage interconnection network. Each processor executes program code written using a set of parallel processing language primitives implemented as C-language libraries. PARSIT dynamically performs reconfiguration of the architecture based on the application program. Useful performance statistics provided by PARSIT can be used by the user to analyze, interpret the system and make necessary decisions.

1 Introduction

Performance evaluation of an application program on different parallel and distributed architectures can be used for the selection of the appropriate architecture for an application. Many attempts have been made to simulate and analyze parallel and distributed systems [1, 3, 6, 8]. These simulators have a specific, or a small architectural domain. Some are limited to a specific machine [1, 8]. Other simulators are restricted to a class of parallel machines [3, 6]. However, none of these tools address the issue of performance analysis of dynamic reconfigurable architectures. We believe that PARSIT is the first tool to address this issue. This work discusses the development of PARSIT - Parallel Algorithm Reconfiguration SIMulation Tool. PARSIT is an extension of our earlier work on simulation languages, performance evaluation, and reconfiguration simulation. Fife, Racherla, and Killian [4] developed DYRECT - a tool for dynamic reconfiguration of multi-computer systems. We have combined a modified version of DYRECT with a Graphical User Interface - based simulator to perform simulation, and

performance analysis of parallel and distributed systems into a unified tool which is presently under development. This simulator is capable of providing architectural mapping of the application program onto the major categories of architectures, such as trees, rings, stars, meshes, and cubes, as well as allowing the user to specify a new or experimental reconfiguration strategy and architecture type [4, 10]. The parallel processing simulation libraries provide the mechanism for the user to run an application program on the system. Performance metrics can then be generated for the application program being simulated. By combining the simulation program with a general-purpose dynamic reconfiguration simulator, we are able to greatly expand the potential target architectures. This improves the ability to compare and contrast different architectures for a given application program.

The paper is organized as follows: Section 2 discusses the system's architecture and PARSIT's design. It also discusses the various components of PARSIT. This section also elaborates on the details of the trace and performance-metrics provided by PARSIT. Section 3 explains the details of running an example application using PARSIT. Section 4 provides conclusions of work carried out and discusses future development.

2 Architecture and Design of PARSIT

Dynamic reconfiguration of multicomputer systems is an important topic in the area of computer architecture. DYRECT [4] is an interactive tool for the simulation of dynamic reconfiguration networks. The use of DYRECT aids in the simulation of reconfiguration algorithms and strategies by providing an interactive graphical tool. This tool can be used by the user to investigate the results of different reconfiguration strategies, using a user-friendly graphical user interface (GUI) built on top of a dynamic reconfiguration simulator. The simulator has several common strategies available directly through a set of library

*Correspondence can also be directed to e-mail: gopalr@cs.uoknor.edu

files. Furthermore, new scenarios may be constructed and analyzed by the simulator. Dynamic architectures were introduced by Kartashev [7] in the early 1970's. A dynamic architecture can be viewed as a black box consisting of processor and memory units with differing computing structures such as multicomputers, arrays, and pipelines. Reconfiguration can be either architectural or fault-tolerant.

PARSIT focuses on architectural reconfiguration of multicomputer systems which is an ensemble of stand-alone processors with distributed memory. PARSIT assumes a supervisory system monitor which decides when and how the reconfiguration occurs. A dynamic reconfiguration can assume a set of configurations during the program execution and can improve the performance of system by switching between them. Rings, for example, are quite useful for pipelined computations and control algorithms while stars and trees are suited for divide-and-conquer algorithms like sorting. PARSIT's supervisor invokes DYRECT to perform dynamic reconfiguration of the architecture. DYRECT uses the Shift Register with Variable Bias (SRVB) scheme proposed by Kartashev [7]. Each processor has a unique node number that is decided a priori by the system monitor. The monitor also broadcasts a bias number to all the nodes. The user can choose the resultant architecture and its parameters. Based on this information, each node performs a local computation to get the node numbers of its neighbors that it would connect to at the time of reconfiguration. Kartashev devised strategies for reconfiguring trees and rings [7]. Biswas designed reconfigurable m-ary trees [12] and Ruskey used transpositions to generate binary trees [11]. Racherla and Radhakrishnan [10] developed parameterizable algorithms for rings and trees. These include reconfiguration algorithms for m-ary trees of variable height and a forest of trees with variable height and branching factor [10].

PARSIT consists of the following modules and input and output: (see Figure 1)

Modules

- **Supervisor Monitor :** This module coordinates the functioning of the whole system.
- **DYRECT :** This module performs dynamic reconfiguration of the nodes to form a new architecture.
- **Reconfiguration Libraries :** The libraries contain methods that perform architectural reconfiguration.

- **Parallel Processing Primitive Libraries :** The C libraries called by the user application program(s).

Input

- **Application Programs :** The C and/or FORTRAN programs that invoke the parallel processing libraries (implemented as C functions). See Appendix A for the C-libraries.

- **System Dependent Information :** This information is hardware and processor related. It includes items like processor speed, link speed, and buffer sizes. In the case of heterogeneous processors, multiple files are used to store this information. The user can also input the initial architecture and the system dependent information graphically.

Output

- **Performance Metrics :** Various metrics such as processor utilization are displayed as graphs and charts.
- **Event Traces :** This is a text file that contains the system snapshot at various events during the simulation.

DYRECT is composed of three major systems (see Figure 2). This modular design allows rapid integration of new reconfiguration strategies, as well as improvements to either the GUI or reconfiguration simulator, without impacting the other parts of the system. Many different reconfiguration strategies, including the ones explained above, are implemented as C libraries that are called by the simulator. The system architecture is assumed to be implemented with a multistage augmented shuffle exchange interconnection network proposed by Srinivas and Biswas [12]. This minimizes the overheads introduced by reconfiguration, both in terms of time and hardware. The supervisor (for dynamic reconfiguration) or the user (for static reconfiguration) choose the initial architecture and its parameters to perform reconfiguration. For example, the supervisor/user can choose to reconfigure to a forest of trees, and can choose pertinent parameters like each tree's branching factor and height. By changing the bias, DYRECT can perform intra-topology reconfiguration. PARSIT and DYRECT were designed as Microsoft Visual Basic applications, running in the MS-DOS environment under Microsoft Windows. The libraries and simulator are written in C, allowing rapid porting to other systems. The only modification would be to recompile the libraries and simulator on the new system, and to pro-

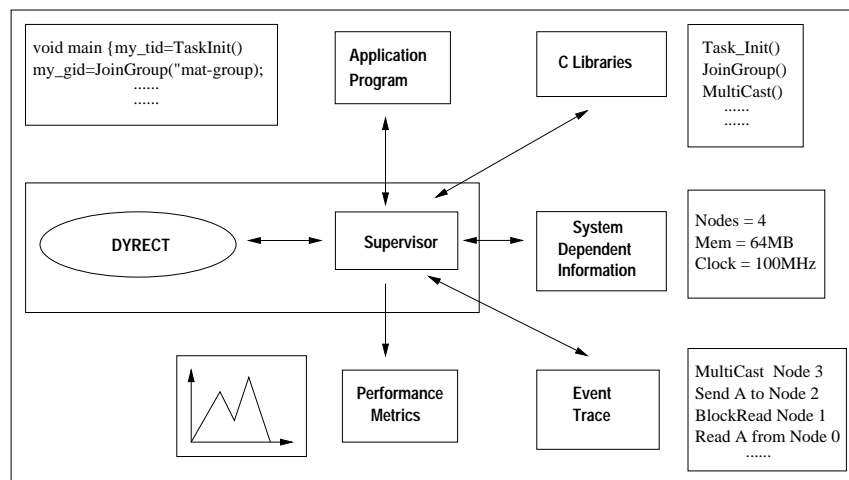


Figure 1: PARSIT Architecture

vide an interface for the host system. As the crux of the work is done within the simulator and the supervisor, changing interfaces will have minimal impact on the tool as a whole.

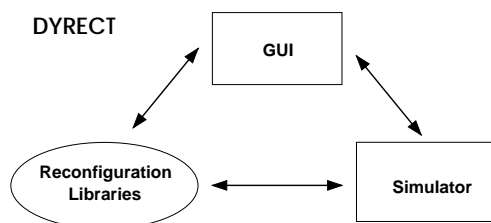


Figure 2: DYRECT Modules

PARSIT's supervisor module performs mapping of the application programs onto processors following reconfiguration. It intelligently performs this mapping by parsing the application program(s) and its associated task-graph (created by PARSIT). It also performs necessary reconfiguration of the architecture to suit the application program. For example, if a portion in the application program involves sorting (using the divide-and-conquer strategy), the supervisor module reconfigures into a forest of trees using DYRECT. This involves making necessary modifications to portions of the application program(s) as may be required to reflect the new architecture. This process is done automatically by the supervisor. The supervisor can also perform load-balancing using process migration.

2.1 Parallel Processing Primitive Libraries

The parallel processing primitives are implemented as C functions. The primitives are based on the PVM C libraries [5] and SIMLANG [1]. The application program is written in C and/or FORTRAN and has embedded function calls to these primitives. The libraries are categorized as Process Control, Group Functions, Static Reconfiguration, Interprocess Communication, and Miscellaneous. Refer to Appendix A for details on the primitives.

2.2 Performance Metrics

Various performance metrics are generated by PARSIT for analysis and use by the user. These include graphs and charts to display processor utilization, distribution of parallelism, system load, status of mailboxes and process queues, process activity graphs and event traces. In addition, other information such as the presence of process and communication bottlenecks is also provided. Work is in progress to animate the execution of the program at various levels of granularity. Figure 3 shows some of these performance metrics in detail.

3 Example Test Case

We show the working of PARSIT using an application program that involves the following subproblems:

- Split-Merge-Sort
- Matrix Multiplication.

The Split-Merge-Sort is best suited for a tree architecture as it exhibits a tree-like process control structure. PARSIT examines the program structure and

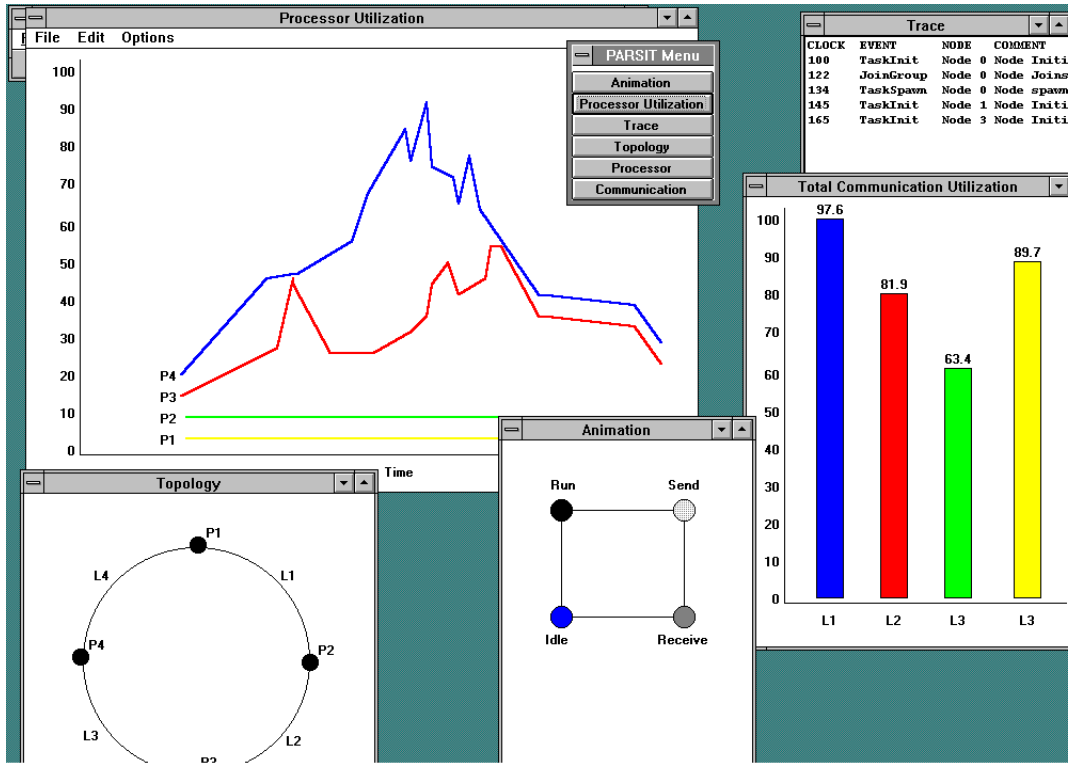


Figure 3: PARSIT Performance Metrics

dynamically reconfigures the architecture as a tree using a reconfiguration algorithm from its library [10].

The matrix multiplication can be done quite efficiently using a hypercube structure. PARSIT intelligently performs the reconfiguration of the architecture to a hypercube using DIRECT. It is noteworthy that the reconfiguration is intelligent and is not explicitly specified by the user. However, the user can statically change the architecture using the Configure, AddHost, and DeleteHost primitives (see Appendix A).

We simulated the working of the above given application program for various number of processors. Appendix B contains portions of the application program and the event trace file. The initial architecture was chosen to be a ring. Figure 3 shows the performance metrics after executing the application program. Figure 4 shows the architectural reconfiguration performed by PARSIT at various stages of the application program.

4 Conclusions and Future Work

We have presented PARSIT, a tool that combines performance evaluation, simulation, and dynamic reconfiguration of parallel and distributed systems. An

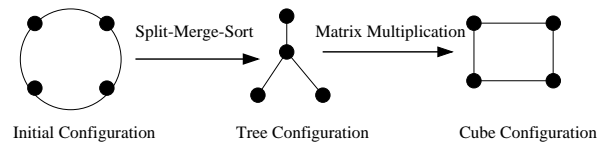


Figure 4: Architectural Reconfiguration

example of how this tool can be used was also discussed. PARSIT provides a valuable tool for studying the effects of dynamic reconfiguration of parallel and distributed processing. Using PARSIT, an application can be represented in the simulation language, and then mapped to a variety of target architectures. The simulation generates performance evaluation metrics such as process utilization and communication utilization. The simulation can also provide analysis of important considerations, such as traffic and process bottlenecks. Using this information, different architectures can be compared to determine appropriate architecture and design decisions for the application program under consideration. Parallel or distributed architectures that are poor choices for a particular application and vice-versa can be determined and avoided.

With this tool, intelligent architecture and design decisions can be made early in the specification and design of an application. Making decisions saves time and other resources during the application's life cycle. Good decisions made early in an application program's life cycle can also improve the quality and usefulness of the finished product.

Future work includes simulating node and link faults as well as incorporating fault-tolerance as part of the system. We are also planning to implement a distributed version of PARSIT on workstations that performs parallel/distributed computing as opposed to simulation (as is presently being done).

Acknowledgements

The authors would like to thank the School of Computer Science at the University of Oklahoma for research funding. Special thanks are due to Prof. Rex Page for his support.

References

- [1] S. Arunkumar, R. Lal, and R. Venkatagopal, "SIMPAC-T: A Simulator For Multitransputer Systems ", *Microprocessing and Microprogramming 35*, 1992, pp. 253-260.
- [2] E.A. Brewer, et. al., "Proteus : A High - Performance Parallel Architecture", **Technical Report**, MIT, June 1991.
- [3] B.B. Bhaumik, P.K. Das, and K.K. Bagchi, "A Simulator For Multi-multiprocessor Systems Design", *Modelling and Simulation*, 1983, pp. 539-544.
- [4] L.D. Fife, G. Racherla, and S.E. Killian, "DYRECT - A Dynamic REConfiguration Tool for Multicomputer Systems.", *Proc. of Workshop on Computer Architecture Education*, in conjunction with HPCA, Raleigh, NC, January 1995.
- [5] A. Geist, et al., **PVM:Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing**, MIT Press 1994.
- [6] I. Glenndinning, et al., "Generic Visualization and Performance Monitoring Tools for Message Passing Parallel Systems", *Proc. of IFIP WG 10.3 Workshop on Programming Environments for Parallel Computing*, Edinburgh, Scotland.
- [7] S.P. Kartashev, and S.I. Kartashev, "Analysis and Synthesis of Dynamic Multicomputer Networks that Reconfigure into Rings, Trees, and Stars",

IEEE Transactions on Computers, Vol. C-36, No. 7, July 1987.

- [8] A.E. Knowles, "PARSIFAL - A Parallel Simulation Facility based on the Transputers", *Workshop on School on High Performance Architectures and Algorithms*, Primisko, Bulgaria, May 1987.
- [9] G. Racherla, R. Parekh, et al., "Simulator for Parallel Architectures and Computing", **Bachelor of Engineering Thesis**, Victoria Jubilee Technical Institute, Bombay, August 1991.
- [10] G. Racherla, and R. Sridhar, "Parametrized Reconfiguration of a General Architecture into Tree Structures and Binary Hypercube", manuscript in preparation.
- [11] F. Ruskey, et al, "Generating Binary Trees by Transpositions", *Journal of Algorithms*, Vol. 11, 1990.
- [12] S. Srinivas, and N.N. Biswas, "Design and Analysis of a Generalized Architecture for Reconfigurable m-ary Tree Structures", *IEEE Transactions on Computers*, Vol. 41, No. 11, November 1992.

Appendix A - Parallel Processing Primitives

Process Control

```
int task_id = TaskInit(void)
int info = TaskEnd(void)
int ntasks = TaskSpawn(char* task_label, int flag,
    char* spawn_data, int num_tasks,
    int** task_list)
```

TaskSpawn flag options:

```
0 = any host can be used
1 = spawn_data holds list of hosts
2 = start with debug set to "on"
```

```
int info = TaskKill(int task_id)
```

Group Functions

```
int info = Sync(char* group_label, int ntasks)
int parent_tid = GetParent(char* task_label)
int group_id = JoinGroup(char* group_label)
int info = LeaveGroup(char* group_label)
int group_size = GetSize(char* group_label)
int task_id = GetTid(char* group_label,
    int group_id)
int group_id = GetGid(char* group_label,
    int task_id)
```

Static Reconfiguration

```
int info = AddHosts(char** host_list,
    int num_hosts)
int info = DelHosts(char** host_list,
    int num_hosts)
int info = Configure(int type)
```

Communication

```
int info = MultiCast(int flag, char* label,
    int** host_list)
int info = BlockRead(int source, int flag,
    void* data, int message_label)
int info = NonBlockRead(int source, int flag,
    void* data, int message_label)
int info = Write(int destination, int flag,
    void* data, int message_label)
```

Read/Write flag options:

```
0 integer 1 char
2 short 3 long
4 float 5 double
```

Miscellaneous

```
void PrintError(int fd, int error_code)
int info = Time(int flag, int total_time,
    int comp_time, int comm_time,
    int wait_time)
```

Appendix B - Example Test Case: Split-Merge-Sort and Matrix Multiplication

```
/* Portion to Perform Split-Merge-Sort */
for(i=1;i <= log2(NumProcs);i++) {
    /*Loop through height of created tree */
    if (my_tid < 2i)
        /*my_tid XOR 2i gives the TID of the
        neighbor node*/
        info=TaskSpawn("treetask",1,
            (my_tid XOR 2i),1,right);
    if (my_tid < 2i-1) {
        /*Split the list*/
        midpoint=Partition(List);
        /*Send half*/
        info=Write(my_tid XOR 2i,
            data_type, List[midpoint+1],100);
    }
    else
        BlockRead(my_tid XOR 2i,
            data_type, List[0],100);
    Size/=2; /*List is now only half as large
}
...
```

/*Portion to perform the matrix-matrix product C = A*B in parallel*/

```
/* Given: TASKS is the number of tasks
SIZE is the size of the matrix M is
SIZE/ROOT(TASKS)
A is the local portion of Matrix A
B is the local portion of Matrix B */
{my_tid=TaskInit(); /*Initialize*/
my_gid=JoinGroup("mat-group");
/*Join Group*/
if (my_gid==0) /*First process in group*/
    info=TaskSpawn("mat-task",0, "nocare",
        TASKS-1,child_list);
    /*Spawn slaves*/
info=Sync("mat-group",TASKS);
/*Force Synchronization*/
for (i=0;i < m;i++)
    my_row[i]=GetTid("mat-group",
        (my_gid/M)*M+i);
    /*Determine Neighbors*/
row=my_gid/M; /*Find row in Grid*/
col=my_gid
up=GetTid("mat-group",((row)?(row-1):
    (M-1))*M+col);
/*Find Above Neighbor*/
down=GetTid("mat-group",((row==(M-1))
    ?col:(row+1)*M+col)); /*And below*/
/*Actual Multiplication*/
for (i=0;i < M;i++) {
    if (col==(row+i)
        info=MultiCast(1,"matgroup",my_row,A,15);
        BlockMult(C,A,B,M);
    }
    else {
        info=BlockRead(GetTid("matgroup",
            row*M+(row+i)
            BlockMult(C,ATMP,B,M);
    }
    info=Write(up,5,B,10);
    info=BlockRead(down,5,B,10);
}
...
return(0);
}
```

Trace Record (Portion)

```
100 TaskInit Node 0 Node Initialization
123 JoinGroup Node 0 Node Joins Group
134 TaskSpawn Node 0 Node spawns slaves
145 TaskInit Node 1 Node Initialization
163 TaskInit Node 3 Node Initialization
```