A Class of Loop Self-Scheduling for Heterogeneous Clusters

A. T. Chronopoulos, R. Andonie, M. Benche and D. Grosu

Department of Computer Science, University of Texas, San Antonio,

We Acknowledge Our Research Sponsors: **NASA** , **The State of Texas Higher Education Coordinating Board**

- Parallel loop styles
- a Master-slave model
- ('Simple') self-scheduling schemes for homogeneous systems
- Distributed self-scheduling scheme
- New (simple) load balancing scheme for homogeneous systems
- Implementation of the simple schemes
- New distributed schemes
- Implementation of distributed schemes
- Tests
- Conclusions

Notation:

- PE is a processor in the parallel or distributed system;
- I is the total number of iterations of a parallel loop;
- p is the number of PEs in the parallel or distributed system;
- P_1, P_2, ..., P_p represent the p PEs in the system;
- A *chunk* is a collection of consecutive iterations. C_i is the chunk-size at the i-th scheduling step (where: $i = 1, 2..$);
- N is the number of scheduling steps;
- t_j, $j = 1, .., p$, is the execution time of P_j to finish all its tasks assigned to it by the scheduling scheme;
- $T_p = \max_{j=1,..,p}(t_j)$, is the parallel execution time of the loop on p PEs;

Example of Parallel Loop:

```
/* increasing */
DOALL K = 1 TO I
      Serial DO J = 1 TO K
                Serial Loop Body
      End Serial DO
END DOALL

/* decreasing */
DOALL K = 1 TO I
      Serial DO J = 1 TO I-K+1
                Serial Loop Body
      End Serial DO
END DOALL
```

Parallel loops distributions

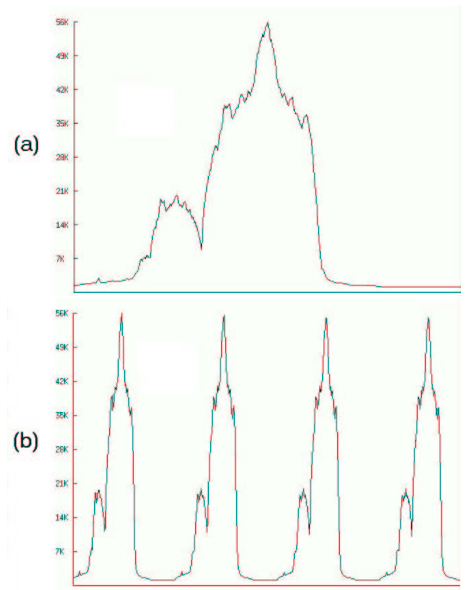

Figure 1: Mandelbrot Set, (a) original and (b) reordered distribution

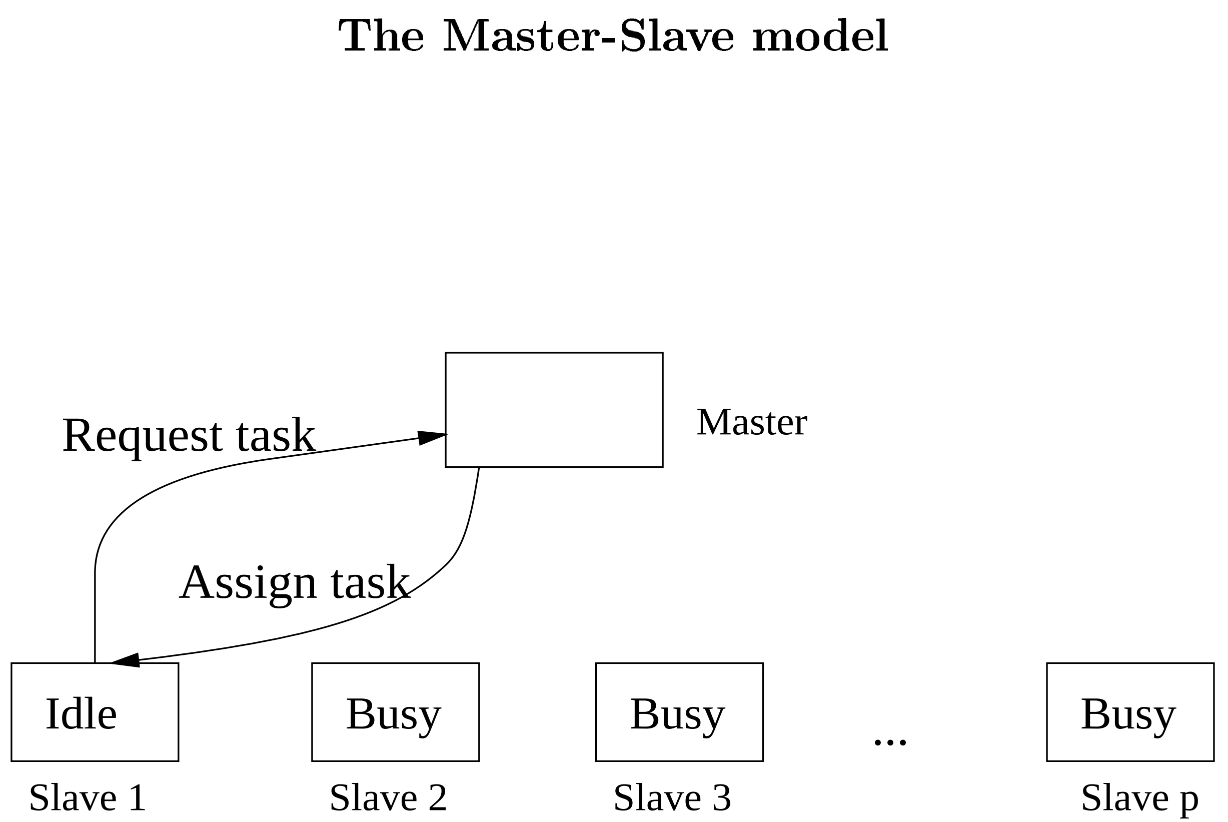

Figure 2: Self-Scheduling schemes: the Master-Slave model

Self Scheduling Schemes

In a generic self-scheduling scheme, at the i-th scheduling step, the master computes the chunk-size C_i and the remaining number of tasks R_i:

$$R_0 = I, \qquad C_i = f(R_{i-1}, p), \qquad R_i = R_{i-1} - C_i \quad (1)$$

where $f(,)$ is a function possibly of more inputs than just R_{i-1} and p. Then the master assigns to a slave PE C_i tasks. Imbalance depends on the (execution time gap) between t_j, for $j = 1, .., p$.

Examples:

Trapezoid Self-Scheduling (TSS) $C_i = C_{i-1} - D$, with (chunk) decrement : $D = \left\lfloor \frac{(F-L)}{(N-1)} \right\rfloor$, where: the first and last chunk-sizes (F,L) are user/compiler-input or $F = \left\lfloor \frac{I}{2p} \right\rfloor$, $L = 1$.

Factoring Self-Scheduling (FSS) $C_i = \lceil R_{i-1}/(\alpha p) \rceil$, where the parameter α is computed (by a probability distribution) or is suboptimally chosen $\alpha = 2$ ([**?**]). The chunk-size is kept the same in each *stage* (in which all PEs are assigned one task) before moving to the next stage. Thus $R_i = R_{i-1} - pC_i$ after each stage.

A new simple Trapezoid scheme with stages

The size of a the next chunk is the sum of the next p chunks that would have been computed by the TSS algorithm. The chunk is then equally divided among the p processors, as in FSS. Thus the $TFSS$ chunk-size is computed:

$$C_j^{TFSS} = \sum_{i=k}^{k+p} C_i^{FSS}$$

Table 1: Sample chunk sizes for $I = 1000$ and $p = 4$

Scheme	Chunk size
S	250 250 250 250
TSS	125 117 109 101 93 85 77 69 61 53 45 37 29 21 13 5
FSS	125 125 125 125 62 62 62 62 32 32 32 32 16 16 16 16 8 8 8 8 4 4 4 4 2 2 2 2 1 1 1 1
$FISS$	50 50 50 50 83 83 83 83 117 117 117 117
$TFSS$	113 113 113 113 81 81 81 81 49 49 49 49 17 17 17 17

Distributed Schemes. Terminology :

- V_i is the virtual power of P_i (e.g. $V_i = 1$ for the slowest PE).
- $V = \Sigma_{i=1}^{p} V_i$ is the total virtual computing power of the cluster.
- Q_i is the number of processes in the run-queue of P_i, reflecting the total load of P_i.
- $A_i = \left\lfloor \frac{V_i}{Q_i} \right\rfloor$ is the available computing power (ACP) of P_i (needed when the loop is executed in non-dedicated mode).
- $A = \Sigma_{i=1}^{p} A_i$ is the total available computing power.

Distributed Trapezoid Self-Scheduling (DTSS)

Master:

1. **(a)** Wait for all workers with $A_i > 0$ to report their A_i; sort A_i in decreasing order and store them in a ACP Status Array(ACPSA). For each A_i place a request in a queue in the sorted order. Calculate A. **(b)** Use $p = A$ to obtain F, L, N, D as in TSS.

2. **(a)** While there are unassigned iterations, if a request arrives, put it in the queue and store the newly received A_i if it is different from the ACPSA entry.
   (b) Pick a request from the queue, assign the next chunk with $C_i = A_i * (F - D * (S_{i-1} + (A_i - 1)/2))$, where: $S_{i-1} = A_1 + .. + A_{i-1}$.

2. **(c)** If more than half of the A_i's changed since the last time, update the ACPSA and go to step 1, with total number of iterations I set equal to the number of remaining iterations.

Slave:

1. Obtain the number of processes in the run-queue Q_i and recalculate A_i. If $(A_i > 0)$ goto step 2.
   else goto step 1.

2. Send a request (containing its A_i) to the coordinator.

3. Wait for a reply; if more tasks arrive
   { compute the new tasks; go to step 1; }
   else terminate.

New distributed self-scheduling schemes:

Modifications of the DTSS algorithm part 1.(b):
(i) $DFTSS$: same as $DTSS$ 1.(b); (ii) $DFSS$: 1.(b) not needed
; *Modifications of the DTSS algorithm part 2.(b)*:
(i) $DFTSS$: 2.(b) Compute $SC_k = \Sigma_{j=1}^{p} C_j^{TSS}$ and C_j^k; (ii) $DFSS$: 2.(b) Compute $SC_k = \left\lfloor \frac{2R_{i-1}}{A} \right\rfloor$ and C_j^k;

Conclusions

Distributed extensions for some important loop self-scheduling schemes were obtained . The main feature of the new schemes is that they take into account the computer processing speeds and their actual loads. Thus the master adapts the assigned load accordingly in order to maintain load balancing. Our test results demonstrate that the new schemes are effective for distributed applications with parallel loops (i.e. loops without inter-iterations dependencies).

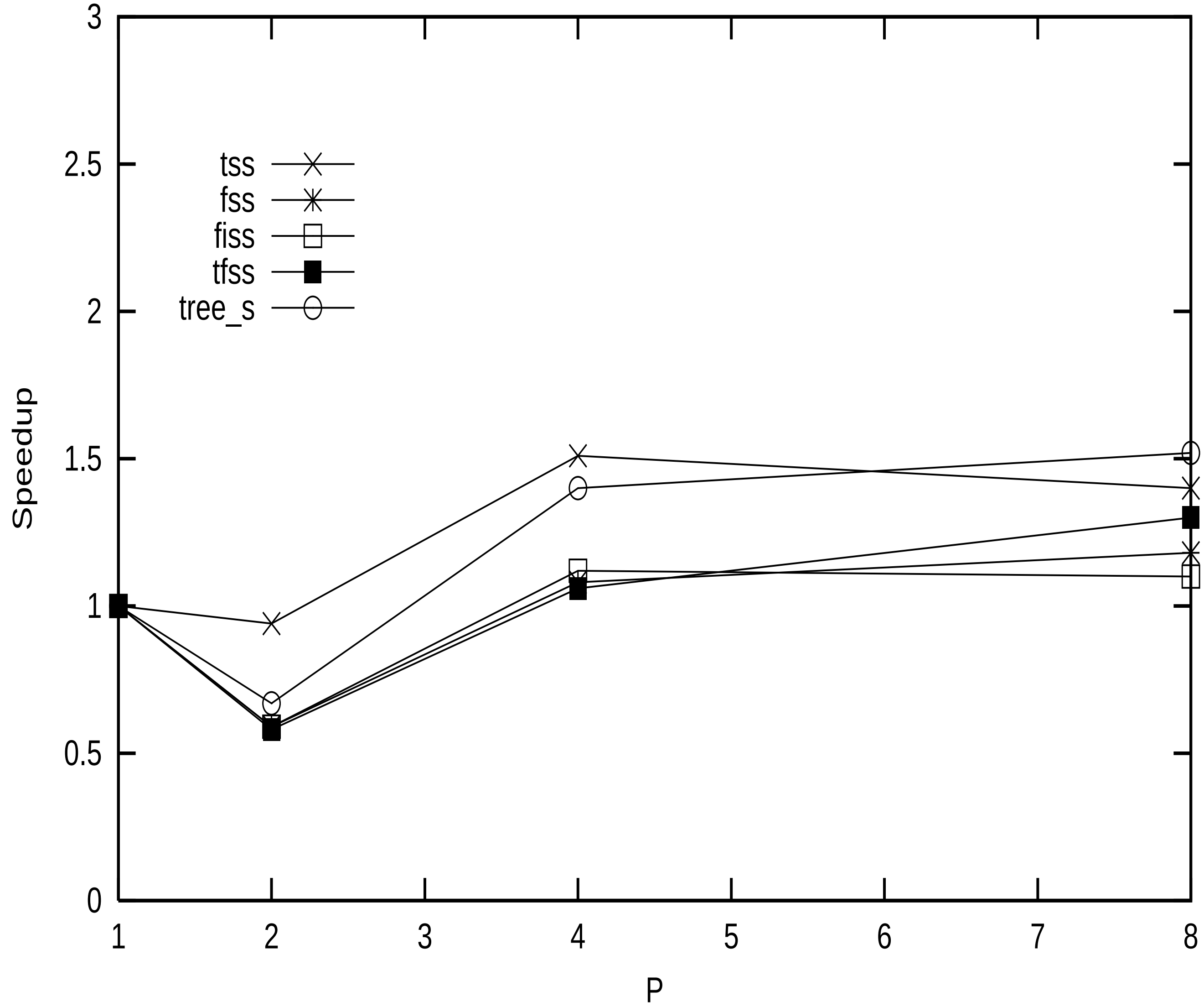

Figure 3: Speedup of Simple Schemes - Dedicated

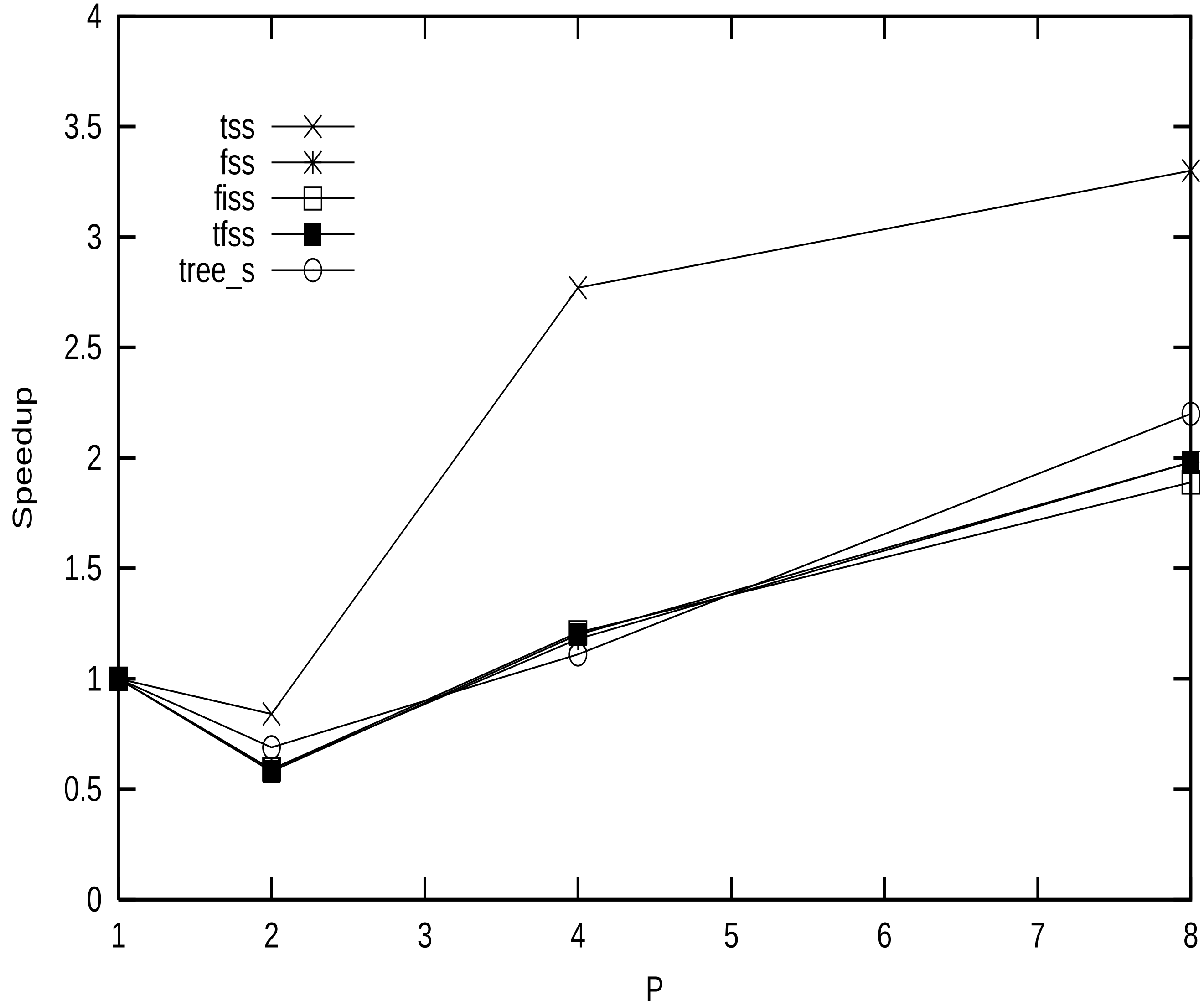

Figure 4: Speedup of Simple Schemes - NonDedicated

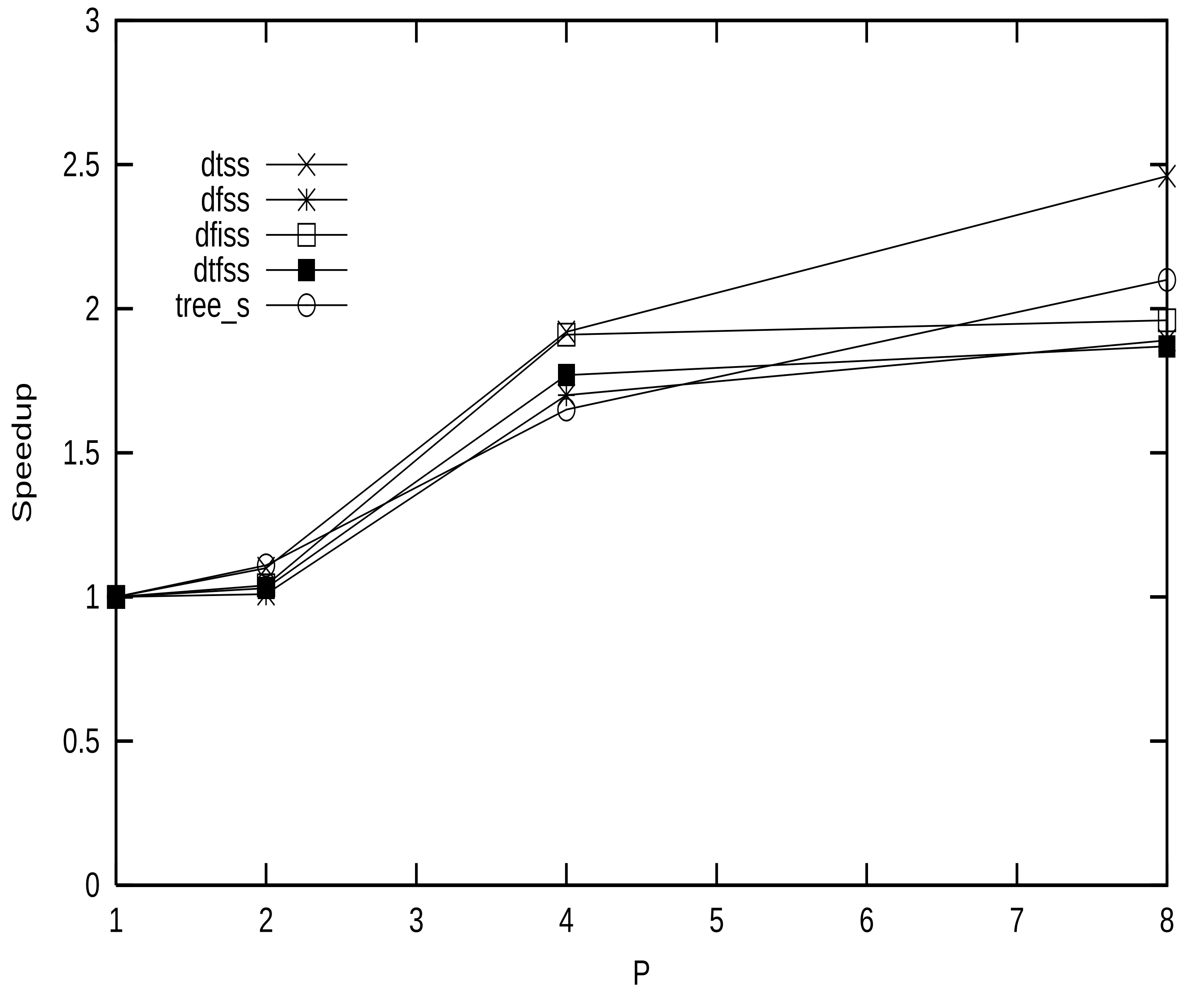

Figure 5: Speedup of Distributed Schemes - Dedicated


6
5
4
3
2
1
0

Table 2: Dedicated Simple Schemes, p = 8; PE_i: $T_{com}/T_{wait}/T_{comp}$ (sec)

PE	TSS	FSS	$FISS$	$TFSS$	$TreeS$
1	2.7/17.5/3.5	0.2/0.8/3.2	1.5/18.5/3.7	2.3/18.7/3.2	0.6/0.0/3.3
2	0.9/18.8/3.7	4.4/15.1/3.3	1.5/19.8/3.4	0.1/0.9/3.2	6.3/12.9/5.4
3	1.3/18.3/3.7	2.8/16.6/3.3	1.8/19.3/3.5	2.1/18.4/3.2	6.1/12.1/5.6
4	1.0/17.5/4.4	1.6/17.6/8.9	1.1/19.8/9.0	0.6/16.7/8.8	6.6/9.2/7.3
5	0.9/12.3/8.0	4.5/9.1/9.3	2.2/7.9/9.6	2.8/9.5/9.7	7.6/6.0/6.5
6	2.4/7.5/10.4	4.2/9.2/9.8	3.8/6.2/8.9	5.0/8.7/9.9	4.9/2.1/9.7
7	4.0/5.7/10.7	3.8/5.9/9.9	3.8/4.4/9.3	4.9/5.9/10.1	3.5/0.0/7.4
8	3.6/4.8/11.8	8.1/4.0/10.4	2.2/4.3/8.9	5.3/4.2/10.2	5.1/0.0/6.0
T_p	23.6	28.1	30.0	26.2	25.0

Table 3: NonDedicated Simple Schemes, p = 8; PE_i: $T_{com}/T_{wait}/T_{comp}$ (sec)

PE	TSS	FSS	$FISS$	$TFSS$	$TreeS$
1	6.0/12.3/9.5	7.0/13.9/9.0	2.3/1.1/9.1	2.9/21.4/11.6	1.8/0.0/9.4
2	8.8/13.4/5.5	0.4/0.8/3.1	2.7/1.0/3.4	2.8/23.0/3.6	9.7/20.8/6.9
3	6.1/14.3/7.2	2.9/2.6/3.4	2.4/1.4/3.5	1.6/18.7/4.0	19.7/19.9/5.7
4	3.0/11.3/13.3	4.6/13.2/28.2	2.8/8.8/27.4	3.9/17.4/26.9	14.0/14.5/15.9
5	4.7/10.4/9.2	9.8/10.7/9.1	3.7/6.0/9.3	3.2/7.4/9.2	23.2/9.0/7.2
6	5.1/8.2/10.3	10.1/10.0/9.6	3.2/7.6/9.1	3.0/5.7/8.9	19.7/4.1/9.8
7	1.1/5.5/17.3	2.8/10.3/28.9	1.5/12.5/27.6	2.7/9.1/28.4	4.6/4.1/15.8
8	4.2/3.8/15.9	6.1/8.9/30.3	2.9/13.7/26.6	3.3/7.1/25.6	15.9/0.0/12.7
T_p	27.8	46.0	48.1	45.8	46.8

Table 4: Dedicated Distributed Schemes, p = 8; PE_i: $T_{com}/T_{wait}/T_{comp}$ (sec)

PE	$DTSS$	$DFSS$	$DFISS$	$DTFSS$	$TreeS$
1	2.2/1.8/6.3	4.1/7.8/5.6	2.5/6.4/5.6	1.4/9.3/5.6	3.1/9.0/5.7
2	2.7/1.2/6.6	2.9/8.8/5.8	2.5/6.2/5.8	1.9/8.5/5.6	3.4/7.7/6.1
3	2.1/1.6/7.0	1.7/10.5/5.3	1.5/7.9/5.4	1.4/9.6/5.6	8.9/0.0/10.2
4	2.5/2.2/5.9	3.3/7.3/6.8	2.3/6.0/6.5	2.4/7.9/6.5	3.1/0.0/5.6
5	2.4/4.2/4.4	2.5/8.3/6.0	1.9/7.3/6.1	1.6/9.2/6.3	2.4/0.0/5.8
6	2.0/5.7/3.7	2.1/8.5/6.3	0.9/9.2/5.6	2.0/9.4/6.0	4.9/0.0/6.1
7	0.5/7.7/4.2	1.6/9.8/5.7	1.9/8.4/5.9	1.5/10.1/6.0	5.2/2.1/5.7
8	1.3/9.5/2.6	2.8/8.3/6.1	3.5/7.3/6.0	3.4/8.3/6.0	4.3/0.0/10.4
T_p	13.4	17.6	16.9	17.6	18.1

Table 5: NonDedicated Distributed Schemes, p = 8; PE_i: $T_{com}/T_{wait}/T_{comp}$ (sec)

PE	$DTSS$	$DFSS$	$DFISS$	$DTFSS$	$TreeS$
1	1.2/3.0/8.5	0.9/9.8/10.6	0.9/8.4/6.5	1.9/9.3/10.7	10.8/14.1/6.7
2	2.2/1.5/8.5	2.5/13.6/7.0	1.6/5.8/7.8	3.4/13.3/6.7	11.1/15.3/6.3
3	1.3/2.4/8.5	2.1/14.7/6.2	0.8/7.3/7.4	1.3/15.7/6.5	7.5/16.2/9.8
4	0.7/5.5/6.6	5.8/0.8/14.4	2.5/3.3/9.8	4.7/2.3/15.1	3.9/0.0/13.6
5	0.4/6.6/7.7	3.0/5.3/13.4	1.3/6.0/8.9	3.2/4.6/14.7	5.9/1.3/14.6
6	1.6/3.9/7.1	0.8/13.7/7.3	1.4/7.3/7.6	0.9/15.6/7.1	9.0/4.0/12.2
7	0.9/8.2/7.2	1.0/8.0/13.0	2.6/5.5/8.4	1.3/7.8/13.1	14.0/7.5/7.0
8	1.8/4.5/6.3	1.1/13.5/7.4	2.1/7.0/8.0	1.8/13.7/7.2	12.6/10.4/8.8
T_p	16.6	23.3	17.7	23.6	33.3