



Optimal Synchronization Frequency for Dynamic Pipelined Computations on Heterogeneous Systems

F. M. Ciorba #, I. Riakiotakis #, T. Andronikos *, A. T. Chronopoulos §, G. Papakonstantinou #

#National Technical University of Athens, Greece

*Ionian University, Corfu - Greece

§University of Texas at San Antonio, TX USA

cflorina@cslab.ece.ntua.gr

1. Introduction

- **Area:** dynamic scheduling of dependent tasks on heterogeneous systems [3] [4]
- **Problem:** determining the optimal synchronization frequency between tasks that minimizes the parallel time
- **Solution:** the theoretical model below, applicable to the class of self-scheduling algorithms (CSS, TSS, FSS, GSS).

2. Background and notations

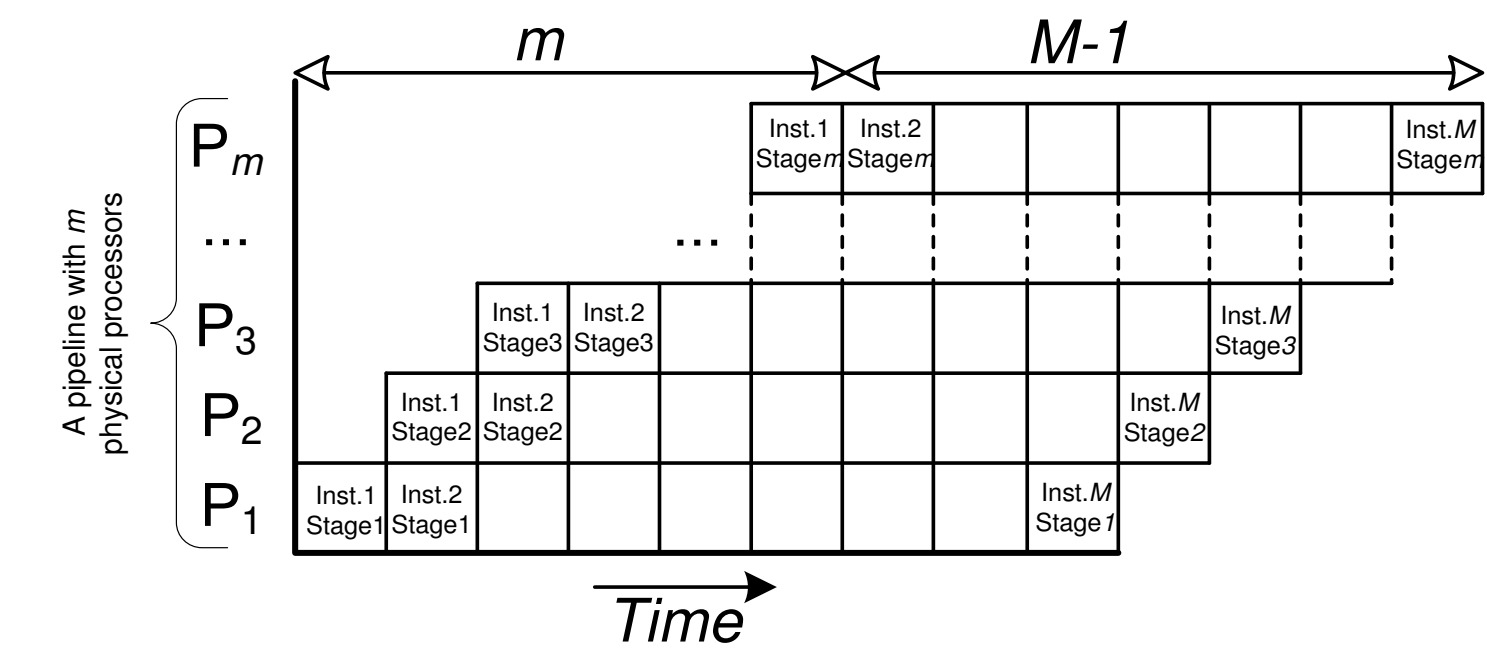


Figure 1: Space diagram of a pipeline with m stages, M instances and $m + (M - 1)$ steps.

- J ($J \subset \mathbb{Z}^n$) - is an n -dimensional Cartesian space, representing the iteration space of a loop nest
- u_1, \dots, u_m - the index points representing the iterations of the loop nest, where $1 \leq u_i \leq U_i$, $1 \leq i \leq n$.
- U_c - the *chunk* dimension, along which the self-scheduling algorithms perform a 1D partitioning of J (see Fig. 2).
- $DS = \{\vec{d}_1, \dots, \vec{d}_r\}$ - the set of dependence vectors (oriented vectors in J).
- We assume a master-worker model with m workers (P_1, \dots, P_m) and one master.
- N - the total number of scheduling steps (i.e., the total number of chunks along U_c).
- V_i - the number of consecutive iterations along U_c assigned to a worker at scheduling step i (also called *chunk size*).
- C_i - the size of the *entire* n -dimensional chunk at scheduling step i .
- U_s - the *synchronization* dimension, along which M synchronization points are inserted (see Fig. 2).
- An assignment round signifies that each worker was assigned a chunk. Each assignment round corresponds to a pipeline with as many stages as the number of workers m (see Fig. 1).

- p - the total number of resulting pipelines (p is different for every self-scheduling algorithm).
- h - the *synchronization interval* (the number of iterations along U_s upon the completion of which neighboring workers synchronize).
- *Synchronization frequency* - the number of synchronization points over the synchronization dimension.
- One pipeline with m stages and M instances requires $m + (M - 1)$ steps to complete.

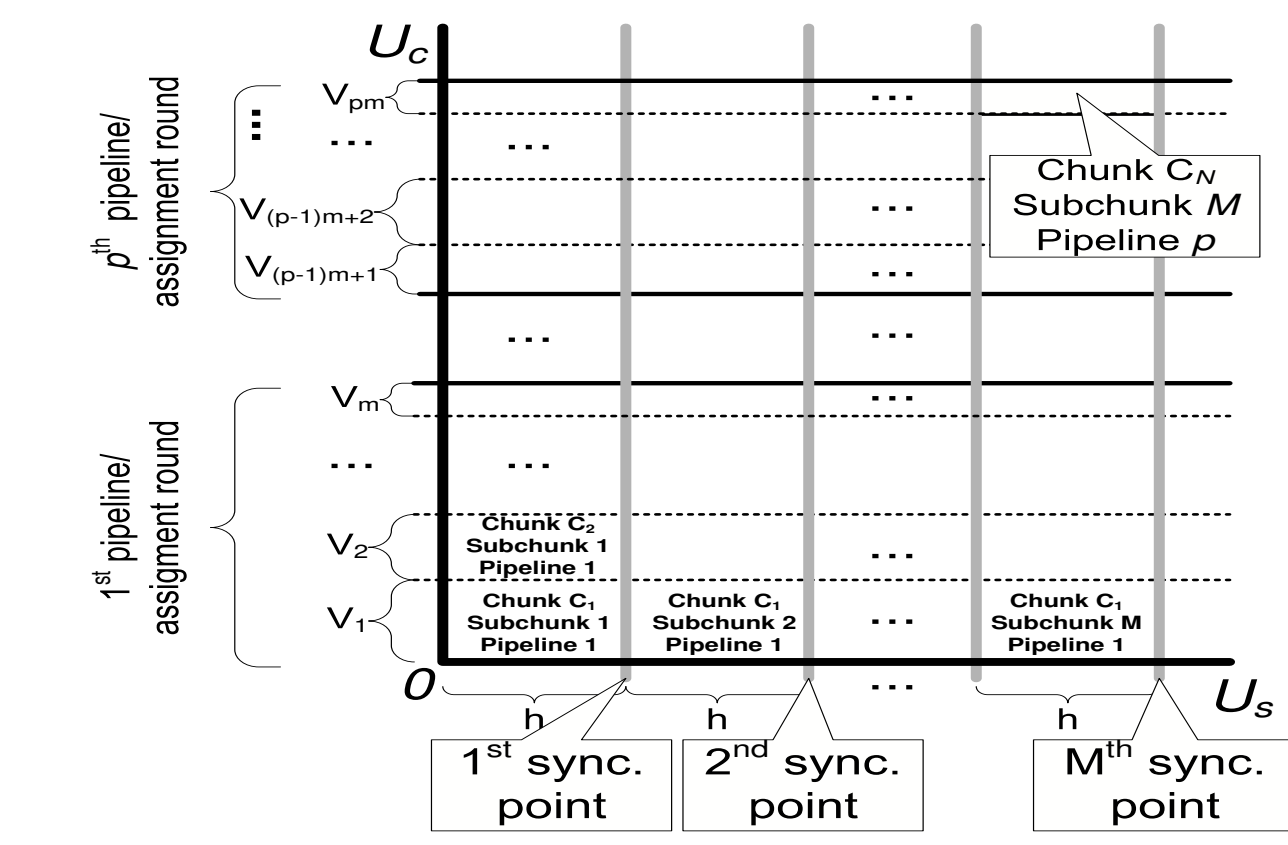


Figure 2: Partitioning of a 2D iteration space into chunks, and placement of synchronization points.

3. Virtualization of heterogeneous systems

- $V P_k$ - the virtual computing power of worker P_k , $k = 1, m$.
- q_k - the number of running processes/jobs in the run-queue of P_k , reflecting its total load.
- $A_k = \frac{V P_k}{q_k}$ - available computing power of worker P_k [1][2].
- $A = \sum_{k=1}^m A_k$ - the total available computing power of the heterogeneous system.
- **Virtualization:** an array of m physical heterogeneous processors is transformed into an array of A virtual homogeneous processors (each with computational power 1), by modeling each heterogeneous physical processor k as a set of A_k virtual homogeneous workers.
- Each assignment round corresponds to a pipeline with A instead of m , and M instances.

Notation: We note the distributed algorithms applied to arrays of A virtual processors as DCSS, DGSS, DTSS and DFSS. Table 1 summarizes the formulas giving the number of scheduling steps and the chunk sizes of the various schemes. F , L , D and α are parameters used by the algorithms and are described in [2].

Table 1: Scheduling steps and chunk sizes

	N	V_i
DCSS	$\frac{U_c}{V_i}$	$1 \leq V_i \leq \frac{U_c}{A}$
DGSS	$A \lceil \frac{U_c}{A} \rceil$	$(1 - \frac{1}{A})^i \frac{U_c}{A}$
DTSS	$\frac{2U_c}{F-1}$	$F - (i-1)D$
DFSS	$A \lceil 1.44 \ln \frac{U_c}{A} \rceil$	$(\frac{1}{\alpha})^{i+1} \frac{U_c}{A}$

4. The proposed theoretical model

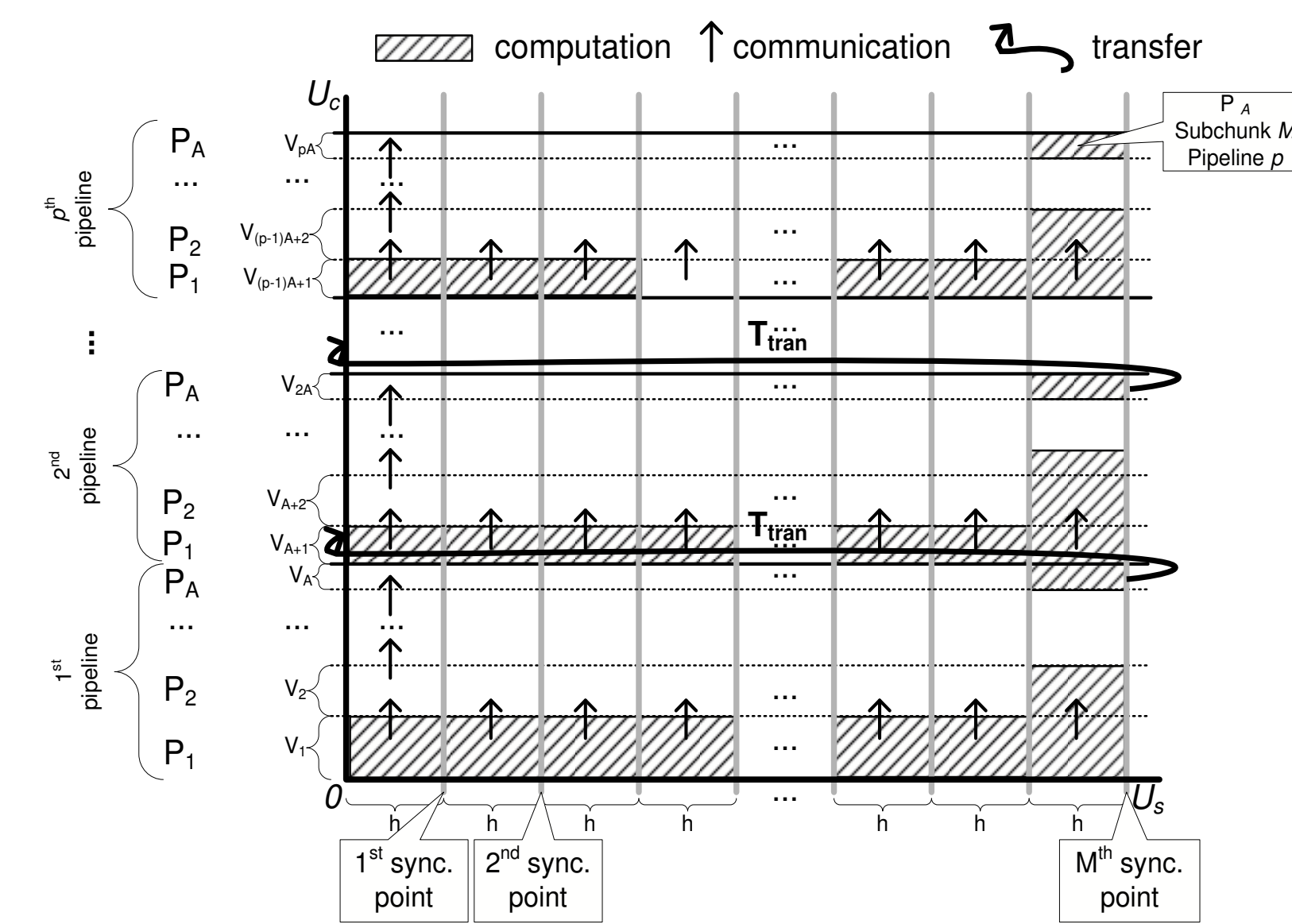


Figure 3: Computation and communication pattern of an iteration space with p pipelines, each with A stages, M instances and $A + (M - 1)$ number of steps.

- **Computation cost model:**
 - Each *virtual processor* has a computational power equal to the computational power of the slowest physical processor.
 - The computation cost of a subchunk is: $t_p = h V_i c_p$, (1) where c_p is the computation time per iteration of the *slowest* worker (Fig. 3), and V_i and C_i are defined above.
- **Communication cost model:**
 - We assume that the cost of sending a message is equal to the cost of receiving a message.
 - The cost t_c of communicating a message of size h between two workers is: $t_c = c_d + h c_c$, (2) where c_d is the start-up cost (the time to send a zero-length message including the hardware/software overhead of sending the message) and c_c is the network throughput, defined as $\frac{1}{\text{sustained bandwidth}}$, where the *sustained bandwidth* is the ratio of the amount of data sent over the actual time measured at the application level.
- **Total parallel execution time:** the completion time of the last subchunk of the problem, illustrated in Fig. 3 as “ P_A , subchunk M , pipeline p ”.
 - We assume that the iteration space is computed in p pipelines, where $p = \lceil \frac{N}{A} \rceil$. The output of one pipeline is the input of the next pipeline.

- Each pipeline can be completed in $A + (M - 1)$ steps (see Fig. 1 and 3). Each of these steps consists of a *receive*, *compute* and *send* operation.
- Thus, the total parallel time is the sum of the completion times of all p pipelines plus the time to transfer data between pipelines.
- The total *computation* time for all pipelines is: $T_{comp} = h U_c c_p \frac{m}{A} + c_p (U_s - h) \sum_{j=0}^{p-1} V_{jA+1}$ (3) where V_{jA+1} is the size of the first chunk for $j = 0, \dots, p - 1$ given by a self-scheduling algorithm.
- The total *communication* time for all pipelines and *transfer* time of all data between any two adjacent pipelines is: $T_{comm} = p(m-2)(2t_c) + p t_c (\frac{U_s}{h} - 1) + (p-1) T_{tran}$ (4) where T_{tran} is the time to transfer the necessary data from one pipeline to the next and is given by $2(c_d + U_s c_c)$ (see Fig. 3).
- Data exchange occurs only between the m physical processors and not between the A virtual processors.
- The *work assignment* time is $T_{wa} = 2t_c + c_{sch}$, where $2t_c$ is the transmission time needed for the work request to reach the master and for the master's reply to reach the worker; c_{sch} is the time needed for the master to compute the next executable chunk size, called *scheduling overhead*.

– The *total parallel time* T_{par} is: $T_{par} = T_{comp} + T_{comm} + T_{wa}$ (5)

Proposition 1 It can be proved that the parallel time T_{par} assumes a minimum as a function of h at:

$$h_{opt} = \sqrt{\frac{U_s p A c_d}{(2m-5) A c_c p + U_c c_p m - c_p A \sum_{j=0}^{p-1} V_{jA+1}}} \quad (6)$$

Table 2: Estimated parameters used to obtain the theoretical time for all experiments.

F-S	c_d	c_c	c_{sch}^{slow}	$V P_{slow}$	$V P_{fast}$
DCSS	$8 \times 10^{-5} s$	$6.55 \times 10^{-7} s$	$6.6 \times 10^{-8} s$	1	2
c_{sch}	$3.2 \times 10^{-5} s$	$3.4 \times 10^{-7} s$	$7 \times 10^{-8} s$	$8.5 \times 10^{-8} s$	$3.7 \times 10^{-8} s$

5. Experimental validation

- Cluster of 17 (16 workers and 1 master) Intel Xeons CPUs at 2.8GHz, with 2GB RAM, interconnected with GigaBit Ethernet.
- Heterogeneous cluster: we put an artificial load in the background of $m/2$ of the workers.
- Test case: Floyd-Steinberg error dithering kernel (computer graphics) on $50K \times 150K$ pixels.

- Parameters estimation: a benchmark program that simulates a small scale master-worker model with cross-node communication. Table 2 summarizes all estimated *computation*, *communication* and *scheduling* parameters.

- Results: Illustrated in Fig. 4 for $h = 20, 40, \dots, 1000$, $m = 16$ and $A = 24$.

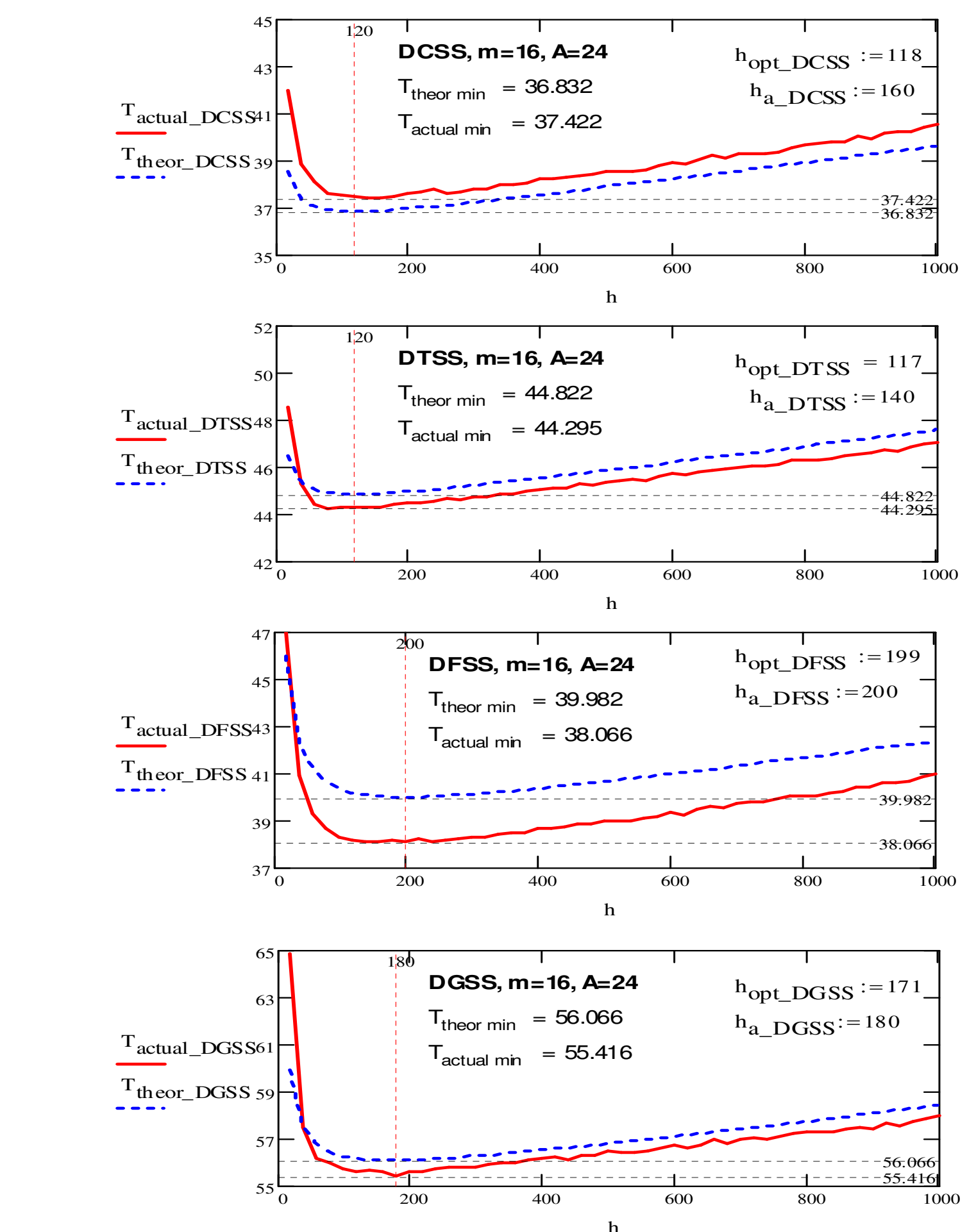


Figure 4: Theoretical versus actual parallel times (sec) and theoretical versus actual h for DCSS, DTSS, DFSS, and DGSS.

References

- [1] T. Kunz. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Trans. on Soft. Eng.* pp. 725–730, 1991.
- [2] A.T. Chronopoulos, S. Penmatsa, J. Xu, and S. Ali. Distributed loop-scheduling schemes for heterogeneous computer systems, *Concurrency and Computation: Practice and Experience*, 18(7):771-785, 2006.
- [3] F.M. Ciorba, T. Andronikos, I. Riakiotakis, A.T. Chronopoulos and G. Papakonstantinou. Dynamic Multi Phase Scheduling for Heterogeneous Clusters, *Proc. of the 20th IEEE Int. Parallel & Distributed Processing Symposium (IPDPS 2006)*, Rhodes, Greece, 2006.
- [4] F. M. Ciorba, I. Riakiotakis, T. Andronikos, G. Papakonstantinou and A. T. Chronopoulos. Enhancing self-scheduling algorithms via synchronization and weighting. *Journal of Parallel and Distributed Computing*, To appear.