Low-complex dynamic programming algorithm for hardware/software partitioning

Divya Ramachandran
Palak Shah

Low-complex dynamic programming algorithm for hardware/software partitioning

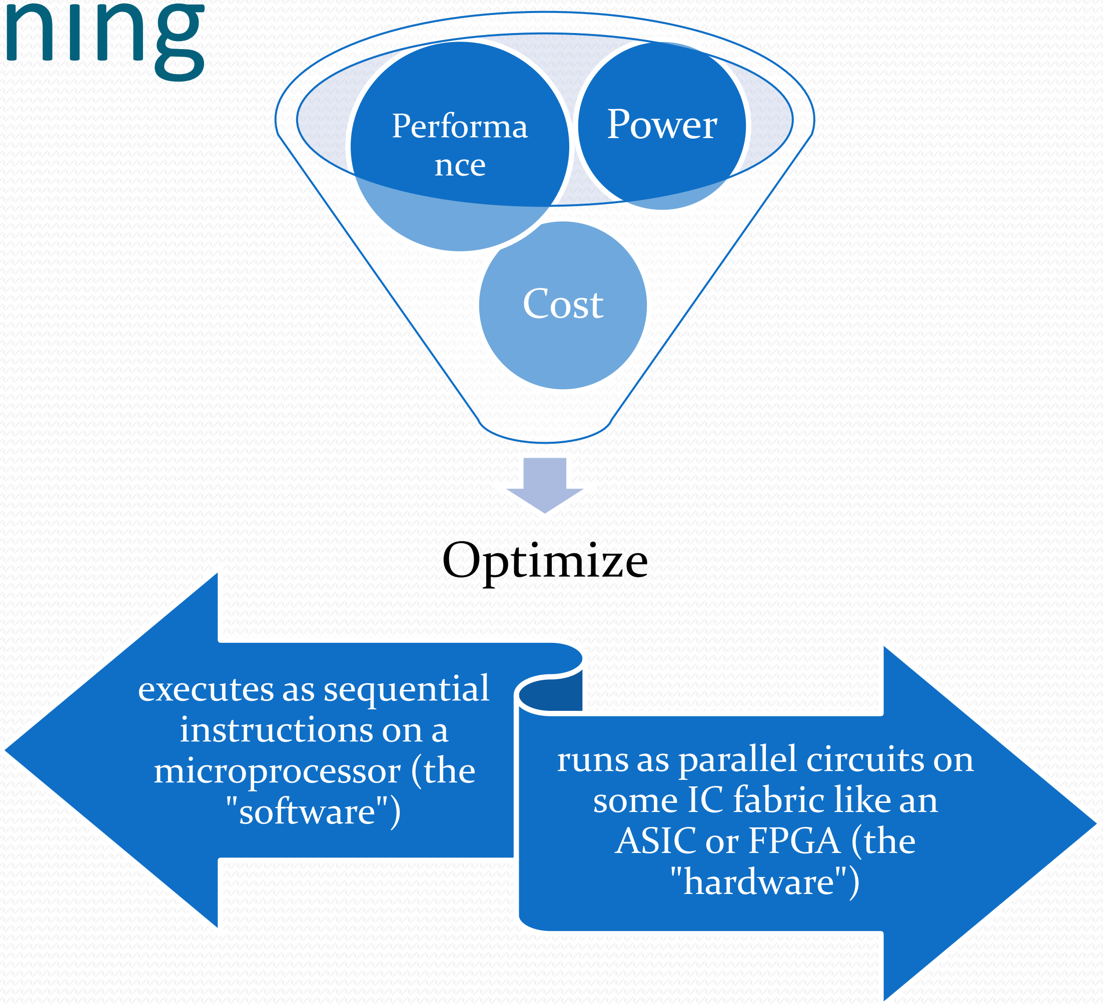
Wu Jigang, Thambipillai Srikanthan

School of Computer Engineering, Nanyang Technological University, Singapore 639798

Received 17 February 2005; received in revised form 1 December 2005; accepted 7 December 2005

Available online 17 January 2006

What is hardware software partitioning


Performa
nce
Power
Cost
Optimize
executes as sequential instructions on a microprocessor (the "software")
runs as parallel circuits on some IC fabric like an ASIC or FPGA (the "hardware")

- The circuit part commonly acts as a coprocessor for the microprocessor

Hardware software partitioning

software

hardware

Can compromise on speed to save cost

Repeated Compute intensive functions

Flexible, cheap

Faster, costlier

Calculator

Running calculations on standard hardware (Excel)

Calculator with a hardware block for every operation

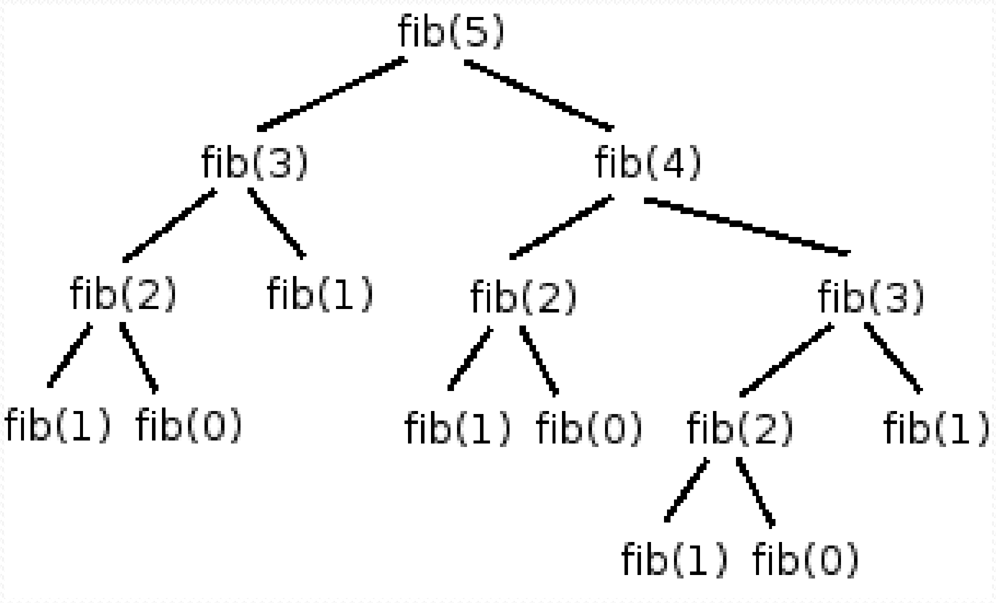
Video compression

Frame handling computations

Fast DCT coprocessor circuit (part of the compression application)

What is the paper about

- An algorithm for partitioning
- Improvement upon an existing algorithm PACE
- Using the concept of dynamic programming :
  - Solving a complex problem by breaking it down into a collection of simpler sub problems and remembering and reusing the earlier solutions

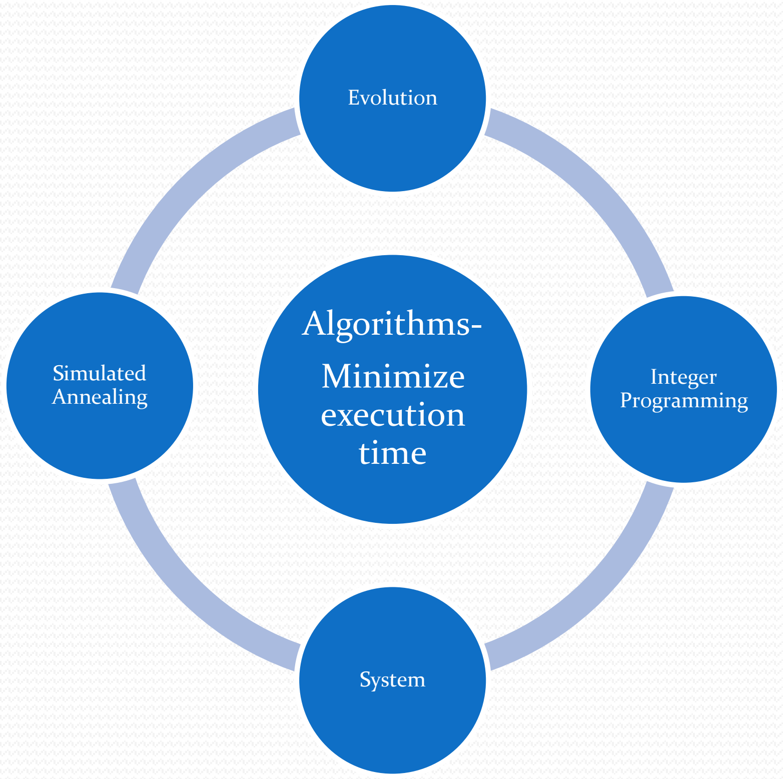


Reference: http://faculty.ycp.edu/~dhovemey/fall2005/cs102/lecture/11-3-2005.html

Approaches

- Everything in hardware → Move parts to software as long as performance constraints fulfilled

- Everything in software → Move parts to hardware as long as time constraint is fulfilled



CDFG

- A CDFG is a set of nodes and directed edges (N, E) where an edge $e_{i,j} = (n_i,n_j)$ from $n_i \in N$ to n_j E N, $i \neq j$, indicates that n_j depends on n_i Because of data dependencies and/or control dependencies
- Divided into basic scheduling code fragments/blocks movable into hardware or software
- Application = $B_1 + B_2 + B_3 \ldots . + B_n$
- The corresponding hardware area, hardware execution time, software execution time and intercommunication delays for each block are provided in advance by a synthesis system

PACE

- Proposed by Knudsen and Madsen
- Employed in the LYCOS co-synthesis system for partitioning control data flow graphs (CDFG)
- Time complexity is $O(n^2 \cdot A)$ and the space complexity is $O(n \cdot A)$ for n code fragments and the available hardware area A

PACE

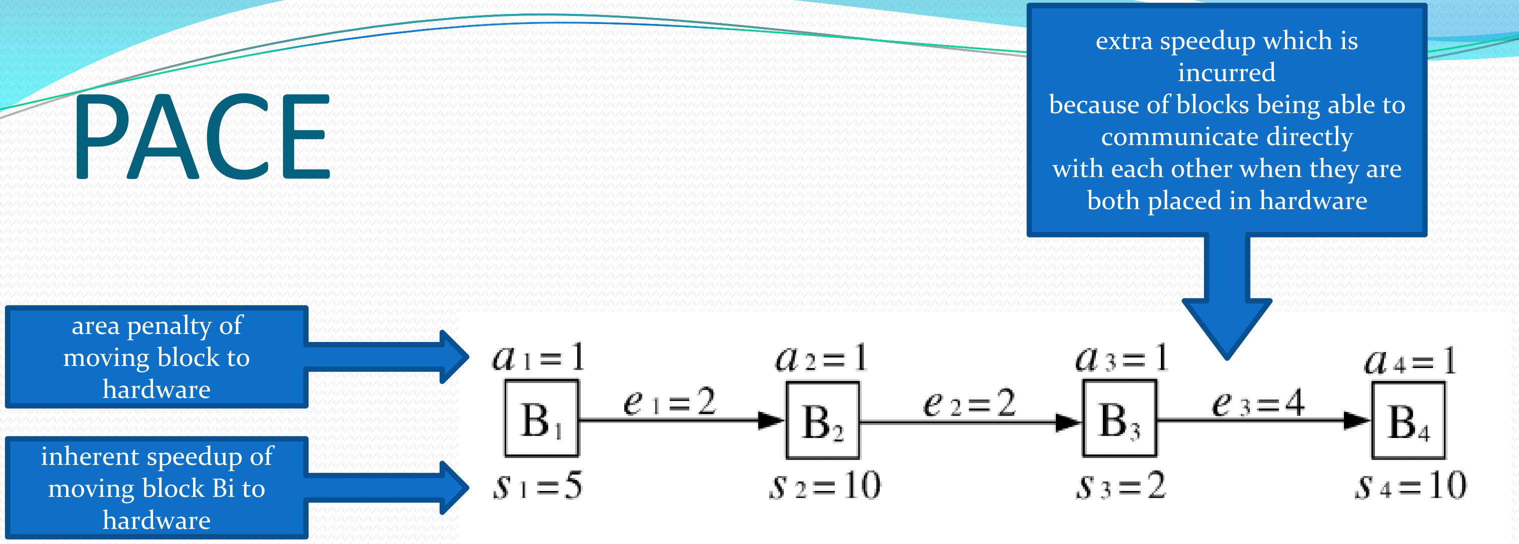
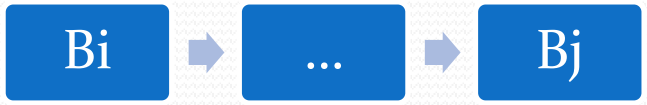

extra speedup which is incurred because of blocks being able to communicate directly with each other when they are both placed in hardware
area penalty of moving block to hardware
inherent speedup of moving block Bi to hardware
$a_1=1$
$a_2=1$
$a_3=1$
$a_4=1$
B_1
B_2
B_3
B_4
$e_1=2$
$e_2=2$
$e_3=4$
$s_1=5$
$s_2=10$
$s_3=2$
$s_4=10$

Fig. 1. Computational model of 4 blocks.

- Hardware blocks and software blocks cannot execute in parallel
- Assumed that the adjacent hardware blocks are able to communicate the read/write variables they have in common directly between them without involving the software side
- Objective is to find the optimal partition to realize the best possible speedup on a given hardware area *A*
- Problem considered in paper is NP-hard

PACE Notations

$S_{i,j}$ where j >= I >= 1



G_j is defined as $\{S_{1,j}, S_{2,j}, \ldots, S_{j,j}\}$, which is called the *j*th group of the sequence
$G_0 \rightarrow$ empty set

Area penalty $a_{i,j}$ of moving $S_{i,j}$ to hardware
= sum of the individual block areas,
i.e., $a_{i,j} = \sum_{k=i}^{j} a_k$

Speedup$(S_{i,j}, a)$ denotes the inherent speedup of
moving $S_{i,j}$ to hardware with available area a

Bestsp(G_j, a) denotes the best speedup achievable by first moving a sequence from G_j to hardware of area a, and then in the remaining area moving a sequence from one of the previous groups, $G_{j-1}, G_{j-2}, \ldots, G_1$, to hardware *Bestsp*$(G_j, a)$ is set to 0 for $G_j = \emptyset$ or $a <= 0$

Bestsp$(G_1 G_2 \cdots G_j, a)$ denotes the best speedup
achievable by moving sequences from $G_1, G_2, \ldots,$
or G_j to hardware of area a

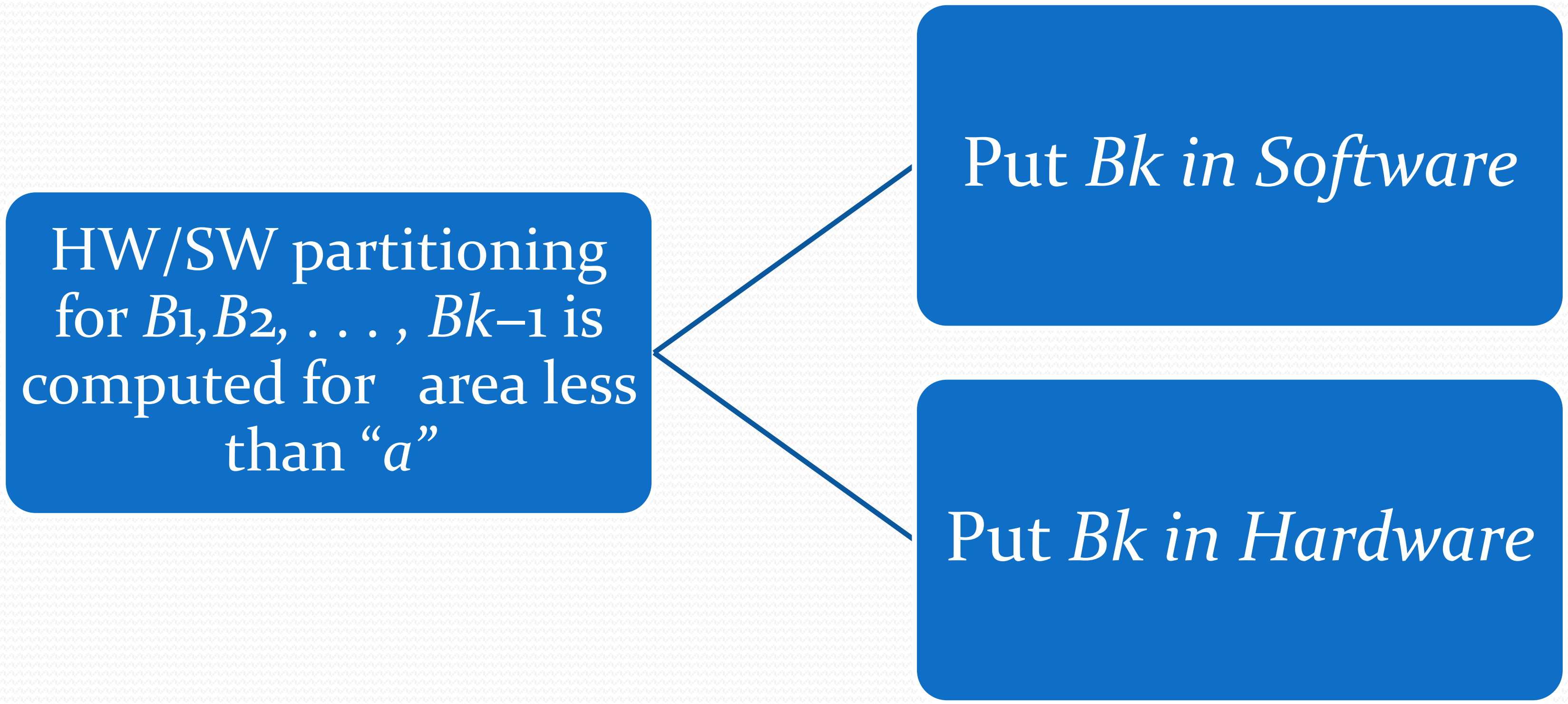
PACE

$$
\text{(A)}\ \begin{cases}
Bestsp(G_j, a) = 0 \quad \text{for } j = 0 \text{ or } a \leqslant 0; \\
speedup(S_{i,j}, a) = \begin{cases} 0 \quad \text{for } a < a_{i,j}; \\ \sum_{k=i}^{j} s_k + \sum_{k=i}^{j-1} e_k \\ \qquad \text{for } a \geqslant a_{i,j}; \end{cases} \\
Bestsp(G_j, a) = \max_{1 \leqslant i \leqslant j} \{speedup(S_{i,j}, a) \\
\qquad\qquad + Bestsp(G_{i-1}, a - a_{i,j})\}; \\
Bestsp(G_1 G_2 \cdots G_j, a) \\
\quad = \max\{Bestsp(G_j, a), Bestsp(G_1 G_2 \cdots G_{j-1}, a)\}; \\
\quad i \leqslant j, \ \ j = 1, 2, \ldots, n.
\end{cases}
$$

- Get partitions for different area values
- We check all parameters for each value
- Time complexity = O(n^2A) if area granularity is 1

SPACE (Simplified PACE)

- Unlike PACE, which relies on a sequence of blocks for computation, SPACE is based on the assignments of only one current block at a time


HW/SW partitioning for $B_1, B_2, \ldots, B_{k-1}$ is computed for area less than "*a*"
Put *Bk in Software*
Put *Bk in Hardware*

SPACE Notations

Bsp(k, a)	• Best speedup achievable by moving some or all the blocks from *B*1,*B*2, . . . , *Bk* to hardware of size *a*
Bsp_sw(k, a)	• Best speedup achievable by keeping *Bk* in software and moving some or all the blocks *B*1,*B*2, . . . , *Bk*−1 to hardware of size *a*. It is clear that *Bsp_sw(k, a)* = *Bsp(k* − 1, *a)*
Bsp_hw(k, a)	• Best speedup achievable by moving *Bk* to hardware and then moving some or all blocks from *B*1,*B*2, . . . , *Bk*−1 to area *a* − *ak*

$$Bsp\_sw(k,a)=0,\qquad Bsp\_hw(k,a)=-\infty,$$
$$Bsp(k,a)=0\quad \text{for } k=0 \text{ or } a=0;$$
$$Bsp\_sw(k,a)=Bsp(k-1,a);$$
$$Bsp\_hw(k,a) = \begin{cases} -\infty & \text{for } a<a_k; \\ \max\left\{ \begin{array}{l} Bsp\_sw(k-1,a-a_k)+s_k, \\ Bsp\_hw(k-1,a-a_k)+s_k+e_{k-1} \end{array} \right\} & \\ \quad \text{for } a \geqslant a_k; & \end{cases}$$
$$Bsp(k,a)=\max\{Bsp\_sw(k,a), Bsp\_hw(k,a)\};$$
$$k=1,2,\ldots,n.$$

Algorithm to explain SPACE

$$Bsp\_hw(k,a)=-\infty,\qquad Bsp(k,a)=0$$
$$\text{for } k\leqslant 0 \text{ or } a=0;$$
$$Bsp\_hw(k,a) = \begin{cases} -\infty & \text{for } a<a_k; \\ \max\left\{ \begin{array}{l} Bsp(k-2,a-a_k)+s_k, \\ Bsp\_hw(k-1,a-a_k)+s_k+e_{k-1} \end{array} \right\} & \\ \quad \text{for } a \geqslant a_k; & \end{cases}$$
$$Bsp(k,a)=\max\{Bsp(k-1,a), Bsp\_hw(k,a)\};$$
$$k=1,2,\ldots,n.$$

Simplified version of Above Algorithm

The best speedup = maximum (*Bsp_sw(k, a)* , *Bsp_hw(k, a)*)

Proposed Theorem

Given n blocks and the list of trial hardware area

A1,A2, . . . ,Am,

both the time complexity and the space complexity of SPACE are $\mathbf{O(n \cdot m)}$*, i.e.,* $\mathbf{O(n \cdot A)}$ *for total hardware area A with granularity of 1*

Simulation and Experimental Setup

- Simulation language : C

- Simulation environment : Intel Pentium-4,
  - 3 GHz,
  - 1 GB RAM.

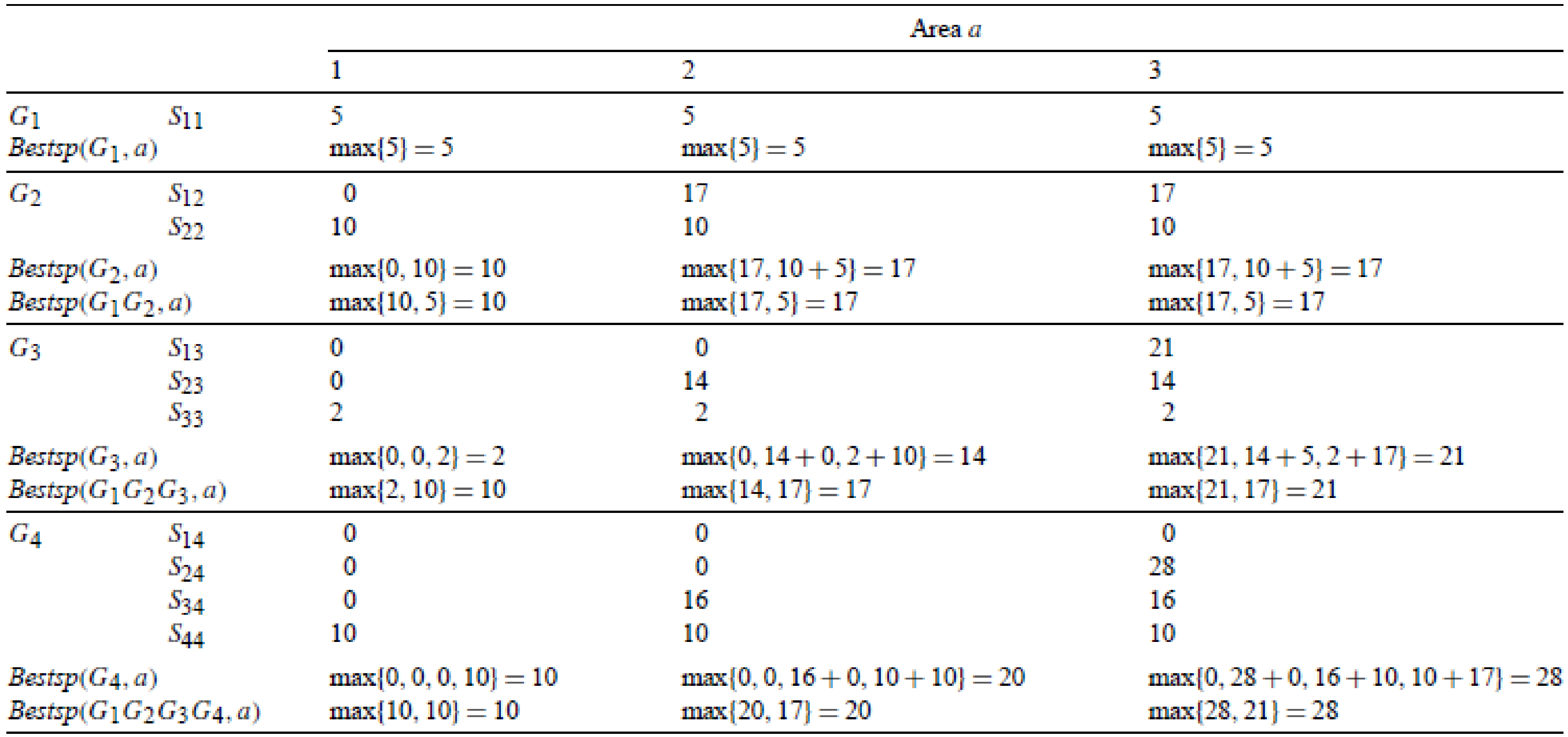
- Variables and constants :
  - For block *Bk*, 1 <= *k* <= *n*, *ak* is randomly generated and satisfies $\sum_{k=1}^{n} ak \leq A$ for a given area *A*.

  - The speedup *sk* and *ek* are randomly generated such that:

    sk = [100, 1000]

    ek = [10, 100]

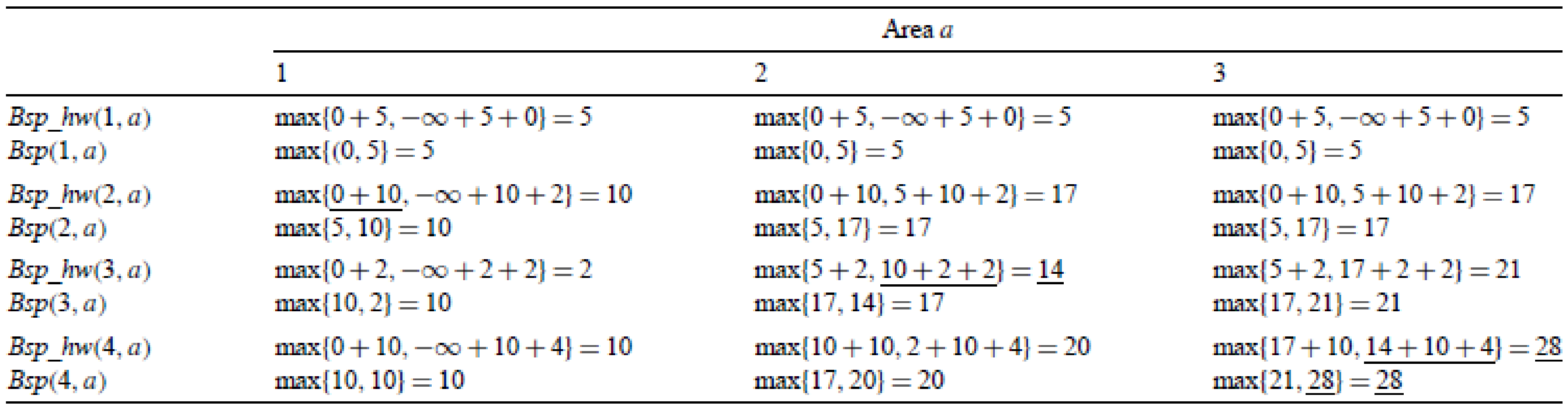
Results – PACE Calculations



		Area a		
		1	2	3
G_1	S_{11}	5	5	5
$Bestsp(G_1, a)$		$\max\{5\} = 5$	$\max\{5\} = 5$	$\max\{5\} = 5$
G_2	S_{12}	0	17	17
	S_{22}	10	10	10
$Bestsp(G_2, a)$		$\max\{0, 10\} = 10$	$\max\{17, 10+5\} = 17$	$\max\{17, 10+5\} = 17$
$Bestsp(G_1G_2, a)$		$\max\{10, 5\} = 10$	$\max\{17, 5\} = 17$	$\max\{17, 5\} = 17$
G_3	S_{13}	0	0	21
	S_{23}	0	14	14
	S_{33}	2	2	2
$Bestsp(G_3, a)$		$\max\{0, 0, 2\} = 2$	$\max\{0, 14+0, 2+10\} = 14$	$\max\{21, 14+5, 2+17\} = 21$
$Bestsp(G_1G_2G_3, a)$		$\max\{2, 10\} = 10$	$\max\{14, 17\} = 17$	$\max\{21, 17\} = 21$
G_4	S_{14}	0	0	0
	S_{24}	0	0	28
	S_{34}	0	16	16
	S_{44}	10	10	10
$Bestsp(G_4, a)$		$\max\{0, 0, 0, 10\} = 10$	$\max\{0, 0, 16+0, 10+10\} = 20$	$\max\{0, 28+0, 16+10, 10+17\} = 28$
$Bestsp(G_1G_2G_3G_4, a)$		$\max\{10, 10\} = 10$	$\max\{20, 17\} = 20$	$\max\{28, 21\} = 28$

Fig. 2. Computations of PACE for the example shown in Fig. 1.

Results – SPACE Calculations

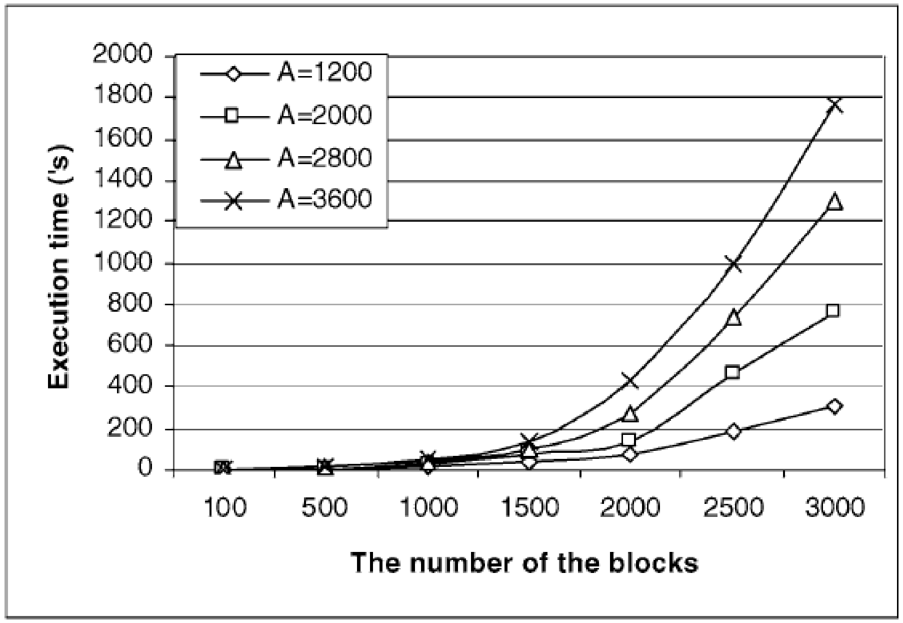


	Area a		
	1	2	3
$Bsp\_hw(1,a)$	$\max\{0+5, -\infty+5+0\} = 5$	$\max\{0+5, -\infty+5+0\} = 5$	$\max\{0+5, -\infty+5+0\} = 5$
$Bsp(1,a)$	$\max\{(0, 5\} = 5$	$\max\{0, 5\} = 5$	$\max\{0, 5\} = 5$
$Bsp\_hw(2,a)$	$\max\{\underline{0+10}, -\infty+10+2\} = 10$	$\max\{0+10, 5+10+2\} = 17$	$\max\{0+10, 5+10+2\} = 17$
$Bsp(2,a)$	$\max\{5, 10\} = 10$	$\max\{5, 17\} = 17$	$\max\{5, 17\} = 17$
$Bsp\_hw(3,a)$	$\max\{0+2, -\infty+2+2\} = 2$	$\max\{5+2, \underline{10+2+2}\} = \underline{14}$	$\max\{5+2, 17+2+2\} = 21$
$Bsp(3,a)$	$\max\{10, 2\} = 10$	$\max\{17, 14\} = 17$	$\max\{17, 21\} = 21$
$Bsp\_hw(4,a)$	$\max\{0+10, -\infty+10+4\} = 10$	$\max\{10+10, 2+10+4\} = 20$	$\max\{17+10, \underline{14+10+4}\} = \underline{28}$
$Bsp(4,a)$	$\max\{10, 10\} = 10$	$\max\{17, 20\} = 20$	$\max\{21, \underline{28}\} = \underline{28}$

Fig. 3. Computations of SPACE for the example shown in Fig. 1.

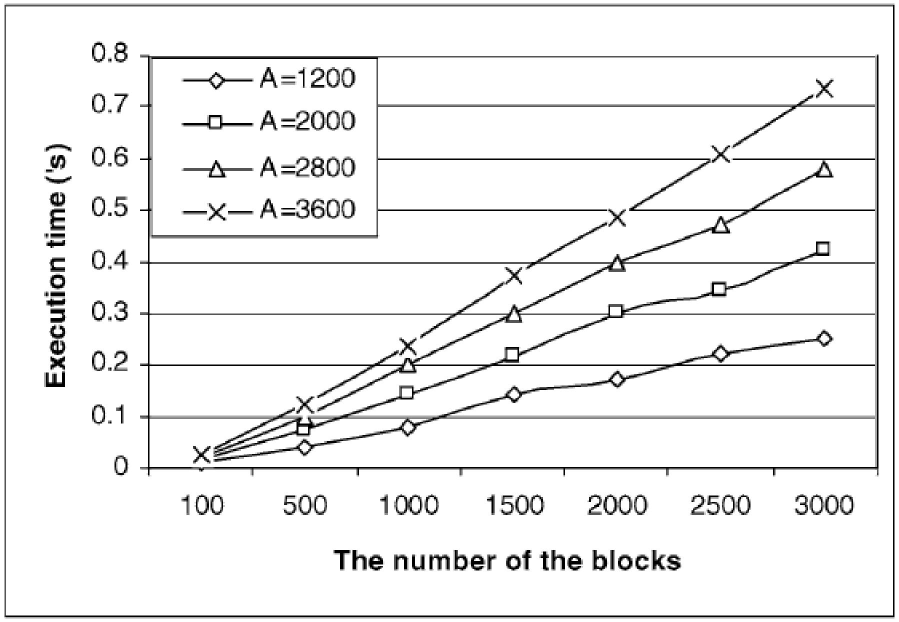
- Max function operates on only two (pre-calculated) values
- Simpler and more elegant way to accelerate the solution

Comparisons in execution time between PACE and SPACE



PACE

O(N²)



O(N)

SPACE

Conclusion

- This paper proposed a new dynamic programming algorithm to accelerate the Hw/Sw partitioning process.

- It is shown that the proposed algorithm is superior to PACE in terms of time complexity. Simulation results confirm that it provides for optimal partitioning even when communication overheads are incorporated.

- It mathematically proves that the time complexity of the latest algorithm is reduced from $O(n^2 \cdot A)$ to $O(n \cdot A)$, without increase in space complexity, where *n* refers to the number of blocks for hardware area *A*.