Performance Results of Running Parallel Applications on the InteGrade

Edson Norberto Cáceres, Henrique Mongelli,
Leonardo Loureiro, Christiane Nishibe
Siang Wun Song

29 de Outubro de 2008

Outline

- Introduction;
- The 0-1 Knapsack Problem;
- Local Alignment Problem;
- Conclusions and Future Work.

The BSP/CGM Model

BSP/CGM model: p of processors, each with its own local memory, communicating through a network.
The algorithm alternates between

- Computation rounds: each processor computes independently.
- Communication rounds: each processor sends/receives data to/from other processors.

Goals:

- Obtain a linear speed-up on p.
- Minimize the number of rounds.

Implementations

- C and C++ languages
- Lam MPI library.
- SPMD paradigm
  - 6 Pentium IV 1.7 MHz nodes
  - 6 AMD Athlon 1.6 MHz nodes
  - 1 GB memory
  - 1 GBit Interconnection Network

Introduction

- Motivation: Classical Combinatorial Problem
  - Wide range of applications;
  - Integer Programming Problem - on constraint.
- Good Algorithms 0-1 Knapsack Problem
  - Integer Programming Research Area.
- NP-Complete Problem
  - Two basic approachs: Dynamic Programming (DP) and Branch-and-Bound (B&B)
- $O(nW)$ time - *Pseudo-Polynomial* - DP
- Our Result: An $O(p)$ communication rounds BSP/CGM algorithm that requires communication with few neighbor processors.

Definition

- *0-1 Knapsack Problem*:
  - $S = \{1, 2, \ldots, n\}$ a set of n distinct items
  - ith item is worth v_i dollars and weighs w_i kilos.
    - v_i and w_i are integers.
  - W is the integer capacity of the knapsack.
- *which items should be selected in order to fill the knapsack with the most valuable load without exceeding the capacity constraint*.

$$\max \left\{ \sum_{i=1}^{n} v_i z_i : \sum_{i=1}^{n} w_i z_i \leq W, z_i \in \{0, 1\} \right\}.$$

The BSP/CGM Algorithm

- Approach: Wavefront - Dynamic Programming - Alves at al.
- p processors, each processor has $O(Wn/p)$ local memory.
- Computing the optimal solution matrix f.
  - $S = \{1, 2, \ldots, n\}$ of items.
  - w, where $w[i]$ is the weight of item i, is broadcasted to all processors
  - $v[i]$ is divided into p pieces, of size $\frac{n}{p}$.
  - Each P_i, $1 \leq i \leq p$, receives the i-th piece of v $(v[(i-1)\frac{n}{p} + 1 \,..\, i\frac{n}{p}])$

The BSP/CGM Algorithm

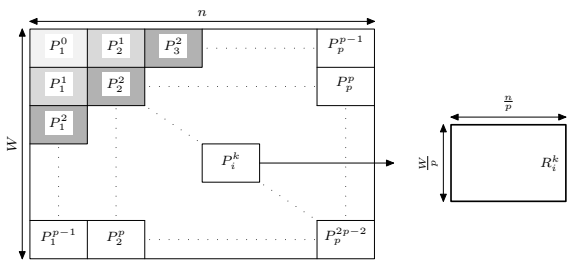
- P_i^k - work of processor P_i at round k.
- P_1 starts computing at round 0.
- P_1 and P_2 can work at round 1.
- P_1, P_2 and P_3 at round 2, and so on.
- After computing f_i^k, P_i sends to P_{i+1} the boundary R_i^k.
- Using R_i^k, P_{i+1} compute f_{i+1}.
- After $p-1$ rounds, P_p receives R_{p-1}^1 and computes f_p^1.
- In the $2p-2$ round, P_p receives R_{p-1}^p and computes f_p^p.
- GOOD, but poor load balancing.

The BSP/CGM Algorithm

Input: (1) The number p of processors; (2) The number i of the processor, where $1 \leq i \leq p$; and (3) The array w, the capacity of the knapsack W and subarray v_i of size $\frac{n}{p}$, respectively.

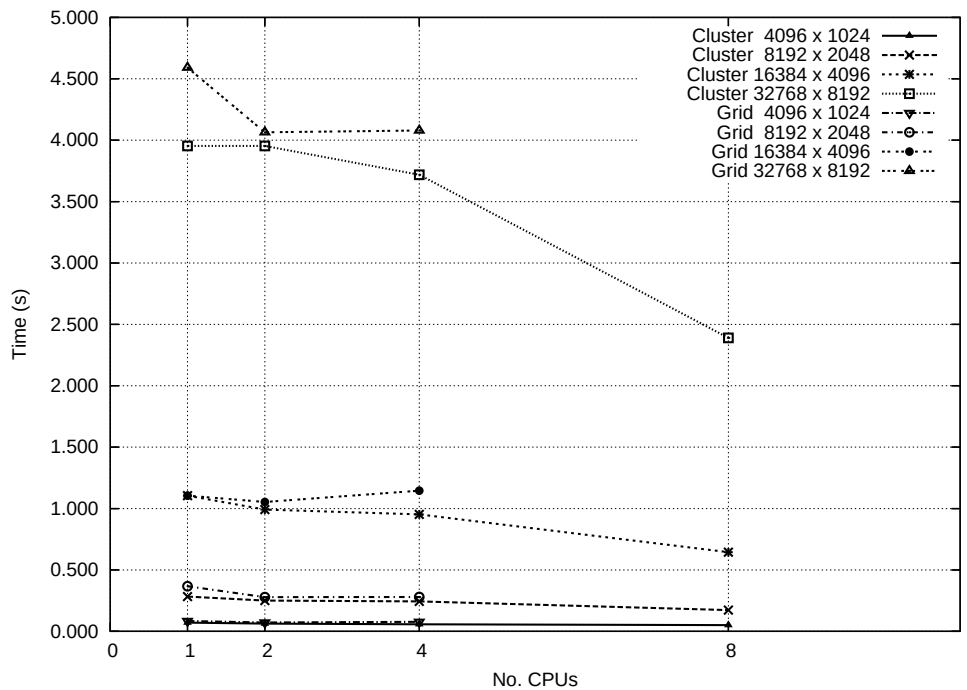
Output: $f(r, c) = \max\{f[r, c - w[r]] + v[r], f[r - 1, c]\}$, where $1 \leq c \leq W$ and $(j-1)\frac{n}{p} + 1 \leq r \leq j\frac{n}{p}$.

for $1 \leq k \leq p$ **do**
  if $i = 1$ **then**
    for $(k-1)\frac{W}{p} + 1 \leq r \leq k\frac{W}{p}$ **and** $1 \leq c \leq \frac{n}{p}$ **do**
      compute $f(r, c)$;
    end for
    send(R_i^k,P_{i+1});
  end if
  if $i \neq 1$ **then**
    receive(R_{i-1}^k, P_{i-1});
    for $(k-1)\frac{W}{p} + 1 \leq r \leq k\frac{W}{p}$ **and** $1 \leq c \leq \frac{n}{p}$ **do**
      compute $f(r, c)$;
    end for
    if $i \neq p$ **then**
      send(R_i^k,P_{i+1});
    end if
  end if
end for



LAM × InteGrade

	4096 × 1024		8192 × 2048		16384 × 4096		32768 × 8192	
p	I	II	I	II	I	II	I	II
1	0.071	0.084	0.283	0.367	1.105	1.105	4.050	4.591
2	0.063	0.072	0.250	0.278	0.992	1.053	3.953	4.065
4	0.057	0.078	0.244	0.280	0.952	1.146	3.718	4.079
8	0.050	-	0.173	-	0.645	-	2.390	-



Introduction

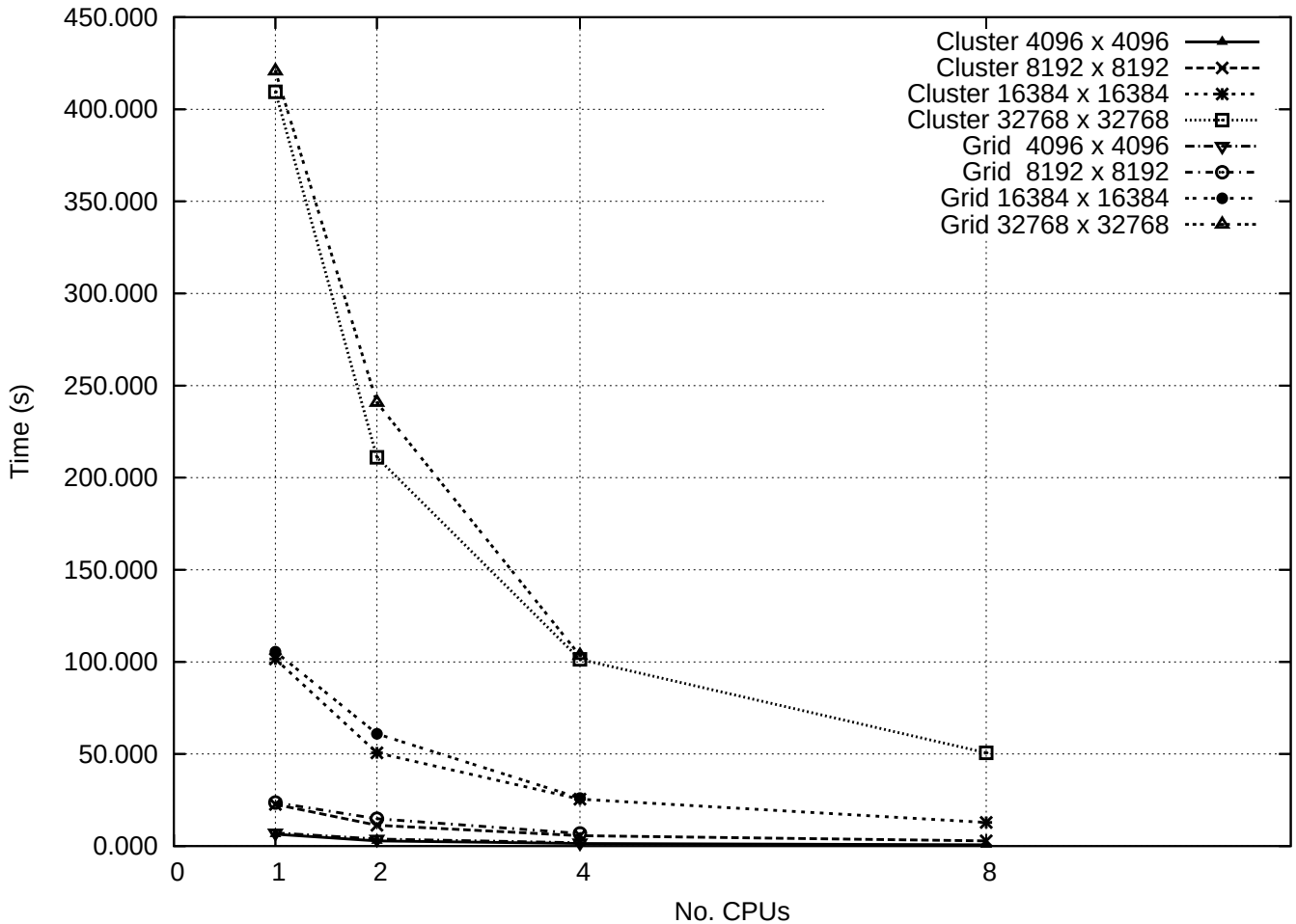
- Motivation: The local alignment is used to determine if two sequences of nucleotides or proteins have similar functionality or evolutionary relationship.
- Basic approach: Dynamic Programming (DP)
- $O(nm)$ time
- Our Result: An $O(p)$ communication rounds and $O(m \times n/p)$ complexity BSP/CGM algorithm (requires communication with few neighbor processors).

The BSP/CGM Algorithm

Input: (1) Sequences S_1 of size m and S_2 of size n; (2) Number of processors p; (3) Rank of processor i; (3) Each processor of rank i holds $s1[0..m-1]$ and $s2[i*(n/p)..(i+1)*(n/p)]$.
Output: Best local alignment between S_1 and S_2

```
matrix A(m+1, blockSize+1), matrix B(m+1, blockSize+1), matrix C(m+1, blockSize+1)
blockSize ← n/p
next ← i + 1
previous ← i − 1
col ← 1
for round ← 0 to p − 1 do
    col ← col + blockSize
    if i ≠ 0 then
        receive (A[0, col..col + blockSize], previous)
        receive (B[0, col..col + blockSize], previous)
        receive (C[0, col..col + blockSize], previous)
    end if
    compute A[1..m, col..col + blockSize]
    compute B[1..m, col..col + blockSize]
    compute C[1..m, col..col + blockSize]
    if i ≠ p − 1 then
        send (A[m, col..col + blockSize], next)
        send (B[m, col..col + blockSize], next)
        send (C[m, col..col + blockSize], next)
    end if
end for
```

LAM × InteGrade



Conclusions

- BSP/CGM Algorithms are suitable for grids.
- BSP/CGM DP Algorithms can be implemented using wavefront strategy.

Future Work

- Fault Tolerant BSP/CGM Algorithms.
- "Balance" the size of the messages.