Parallel Processing in Mixed Integer Programming

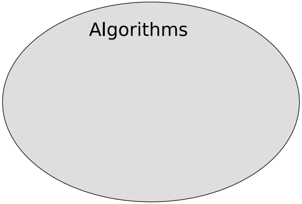
Laurent Poirrier

Université de Liège
Montefiore Institute

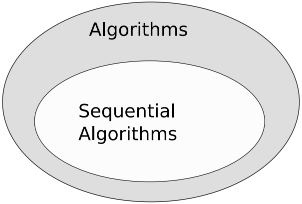
March 27, 2009

Outline

What is Parallel Processing?



What is Parallel Processing?



- Any non-sequential processing permits simultaneous/concurrent computations
- Covers a broad class of very different techniques

Why Parallel?

- Frequency-scaling limits vs. Moore's law
- Computing power is becoming more affordable

How?

- having multiple problems to solve
- non-sequential algorighms (or parts)
- parallel processing techniques

Parallel Computing Resources

Lower level	1- SIMD and Vector Instructions
	2- Many Cores and Specialized Parallel Processing Units
	3- Multiple Cores/CPUs: Multithreading
Higher level	4- Distributed computing: grids and clusters

- low level: more hardware-specific, small operations
- high level: more problem-specific, large groups of operations

For MIP?

Which is most suited for Mathematical Programming?

1- SIMD and Vector Instructions

- Specialized CPU instruction that operate on vectors of numbers.

- No branching possible. Only deterministic vector processing.

1- SIMD and Vector Instructions : Examples

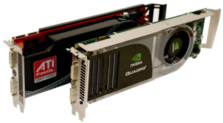
From ffmpeg, SSE code for parts of the FFT (299 lines)

```
        /* do the pass 0 butterfly */
movaps  (%0,%1), %%xmm0
movaps  %%xmm0, %%xmm1
shufps  $0x4E, %%xmm0, %%xmm0
xorps   %%xmm4, %%xmm1
addps   %%xmm1, %%xmm0
```

From mplayer-1.0rc2, SSE2 code for IDCT (839 lines)

```
psrldq_i2r(6, xmm0);      /* xmm0 = - - - 66 65 - - - */
pslldq_i2r(4, xmm5);      /* xmm5 = - - - - 54 51 - - */
por_r2r(xmm1, xmm4);      /* xmm4 = 76 73 67 64 - - 35 33 */
por_r2r(xmm2, xmm3);      /* xmm3 = 77 75 74 - - 53 52 34 */
```

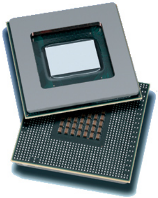
2- Specialized Parallel Processing Units




```
const vec3 a1 = vec3(1., 0., 0.);
const vec3 a2 = vec3(0., 1., 0.);
const float thresh = 0.98;
vec3 nn = normalize(gl_Normal);
vec3 aa1;
if (dot(a1, nn) >= thresh) {
        aa1 = a2;
} else {
        aa1 = a1;
}
vec3 s = normalize(cross(aa1, nn));
vec3 t = normalize(cross(s, nn));
```

GPGPU: General Purpose Graphics Processing Unit

3- Multithreading



- takes advantage of multi-core architectures
- memory is shared

Limitations:
- thread creation overhead
- number of cores

Typically:
threads: $O(8)$; operations: > 1ms.

4- Distributed Computing

- supercomputers and Beowulfs
- memory is not shared

Limitations:
- process creation overhead
- number of computers
- data transfers

Typically:
threads: $O(100)$; operations: > 1s.

4- Distributed Computing: Networking concepts

- propagation delay: independent from the amount of data, related to network devices and topology
- transmission delay: proportional to the amount of data, related to link capacity (max flow)

Scheduling

Clusters are asymetric. What is our objective?

- max/min CPU usage?
- max efficiency in CPU usage?
- fastest solution?

How do we handle failures? And variable resources? Do we have preemption?

Parallel speedup

Let p be the number of processing units,

$$S_p = \frac{T_1}{T_p}$$

Can we achieve linear speedup $S_p = p$?

Specialized techniques

When a given algorithm is intrinsically sequential, what can be done?

Specialized techniques: Racing

Specialized techniques: Lookahead

Example in nonlinear programming:

Increment $k = \{1, 2, \ldots\}$ until $e(k) \leq e_0 = 2.4.10^{-4}$

$$\begin{array}{rcccccc} k = & 1 & 2 & 3 & 4 & 5 & \ldots \\ e(k) = & 4.4.10^{4} & 1.2.10^{-1} & 2.1.10^{-4} & 5.2.10^{-7} & 2.3.10^{-8} & \ldots \\ & & & \downarrow & & & \\ & & & \text{exit} & & & \end{array}$$

In Mixed Integer Programming

- Linear Programming

- Branch and Bound

  - Heuristic cut generation
  - Choice of branching direction
  - Tree backtracking

Linear Programming

Both the simplex algorithm and interior-point methods are iterative. Iterations consists in relatively fast matrix operations. In MIP, solution of LP relaxations is considered fast. But the number of subproblems to solve can be huge.

Branch and Bound: Strong Branching

For every element in the subset of potentially "good" branching directions, a subproblem is created, and a few simplex iterations are performed.

The present, the future

Cplex, gurobi, mosek, coin-or can be multithreaded (mostly using parallel strong branching and racing), but not distributed.

Distributed computing could be explored for backtracking in short term.

Open problem: Scheduling in the case context branch and bound (both for multithread and distributed)

Nonlinear optimization

Given $\mathbf{f}(\mathbf{p})$, find $\mathbf{p}$ such that

$$\mathbf{y} \approx \mathbf{f}(\mathbf{p}) \tag{1}$$

We look for a local minimum of

$$e(\mathbf{p}) = \sum_{i=0}^{M-1} \left(f_i(\mathbf{p}) - y_i\right)^2 \tag{2}$$

whose basin of attraction contains the initial condition

$$\mathbf{p} = \mathbf{p}_0$$

Levenberg-Marquardt

$$\mathbf{q}_i = \left(J^T J + \lambda \,\mathrm{diag}(J^T J)\right)^{-1} J^T \left(\mathbf{y} - \mathbf{f}(\mathbf{p_i})\right) \tag{3}$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{q}_i \tag{4}$$

Levenberg-Marquardt Iteration Scheme (1)

$e_0 = \infty$
$\lambda_0 = 0.001$
$\nu = 10$
$i = 0$

while (**not** stop condition) {

(a): Computation of Jacobian
Computation of terms of $\mathbf{q}$

(b): $e_1 = \text{residue}(\mathbf{p}_i + \mathbf{q}(\lambda_i))$
$e_2 = \text{residue}(\mathbf{p}_i + \mathbf{q}(\lambda_i/\nu))$

if $(e_1 < e_0)$ **and** $(e_1 < e_2)$ {
$\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{q}(\lambda_i)$
$\lambda_{i+1} = \lambda_i$

Levenberg-Marquardt Iteration Scheme (2)

(c):

$$
\begin{aligned}
&\} \textbf{ else if } (e_2 < e_0) \textbf{ and } (e_2 < e_1) \; \{\\
&\qquad \mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{q}(\lambda_i/\nu)\\
&\qquad \lambda_{i+1} = \lambda_i/\nu\\
&\} \textbf{ else if } (e_1 > e_0) \textbf{ and } (e_2 > e_0) \; \{\\
&\qquad e_3 = \infty\\
&\qquad k = 0\\
&\qquad \textbf{while } (e_3 \geq e_0) \; \{\\
&\qquad\qquad e_3 = \text{residue}(\mathbf{p}_i + \mathbf{q}(\lambda_i.\nu^k))\\
&\qquad\qquad \text{k} = \text{k} + 1\\
&\\
&\qquad\qquad \textbf{if } (k \geq \text{bound on } k)\\
&\qquad\qquad\qquad \textbf{exit}\\
&\qquad \}\\
&\qquad \mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{q}(\lambda_i.\nu^k)\\
&\qquad \lambda_{i+1} = \lambda_i.\nu^k\\
&\}\\
&i = i + 1\\
\}
\end{aligned}
$$