

Parallelization of the ORNL Stellarator Optimization Code

S. P. Hirshman (ORNL)

M. Zarnstorff, S. Ethier (PPPL)

A. Ware (UMT)

Introduction

- Stellarator optimization can take a **long** time (even on supercomputers)!
 - Several minutes for each evaluation of the chisq physics target (cost) function
 - Several 100's of iterations to generate Jacobian approximation, perform descent

Parallelization - possibility for significant reduction of clock time

- Levenberg-Marquardt optimizer splits “naturally” into multiple “processes”:
High Level Parallelization possible
- Avoids need for tedious loop optimization of multiple, individual codes
- CPU usage limited only by maximum memory requirements of physics code(s)

Parallelization Possibilities (cont'd)

- Superscalar convergence improvements to Levenberg-Marquardt
 - “optimize” choice of LM parameter after each Jacobian evaluation
 - possible with no additional cost on parallel machine (little overhead on serial machine)
 - more than factor of N (#cpus) speed-up

Structure of Levenberg-Marquardt Optimizer

- Calculate approximate “Hessian” matrix elements *numerically* for least-squares physics target
 - do $l = 1, N$ $!!N = \text{independent vars}$
 $\text{chi_sq} = \text{Chi_Sq}[\text{xb} + \delta x(l)]$ $!!\text{xb}: \text{bdy coeffs}$
 end do
 - form $H_{ij} = d(\text{chi_sq})/dx_i * d(\text{chi-sq})/dx_j$

Levenberg Optimizer (cont'd)

- Determine LM parameter based on Hessian
 - “weight” between steepest descent, Newton
 - estimate several values (can significantly improve convergence, initially at least)
- Repeat until convergence criterion satisfied

Choice of Parallelization Method

- OpenMP Protocol
 - available on some shared memory systems
 - easy to use syntax for loop parallelization
 - won't work on distributed memory systems (CRAY T3E, for example)

Parallelization methods (cont'd)

- MPI (message passing interface)
 - Available on T3E (distributed memory system), J90, RISC (?)
 - restructuring of code generally needed
- Fork
 - short C-code works on any UNIX platform
 - easy interface to existing FORTRAN codes

Fork implementation

- “Parent” process spawns a child or several children
 - `pid = fork();` (<0 if process unsuccessful)
 `if (pid == 0) &funct();`
 `num_processes++;`
- `wait()` used to monitor completion of child processes
 - Jacobian requires *all* processes to finish

“Minor” Code Modification to Implement Fork

- In FORTRAN code, eliminate loop containing function call and “wrap” it
 - call `multiple_process(N, Wrapper(j,f), funct)`
funct is the `Chi_sq` function
Wrapper: calls `f=funct` for `j`'th process, loads commons, writes output array information to file for access outside of multi-process function (can't use commons: overwritten!)

Available Processors

- Chi_sq function implemented using “system” calls (lack of source code, smaller executable)
 - call `system(“xvmec input.filename”)`
- System call itself is implemented by fork
- Limits $N_{\text{processes}} < N_{\text{processors}}/2$

Sample Output (on CRAY/J90)

- N = 8 processors requested
- wall clock time reduced by 8/step
- Begin multi-processing, requesting 8 processes
 - Chi-sq = 3681.93 for iteration 14 (LM parameter = 1.08E-16)
 - Chi-sq = 47086.60 for iteration 13 (LM parameter = 3.43E-16)
 - Chi-sq = 34244.43 for iteration 9 (LM parameter = 1.08E-15)*
 - Chi-sq = 5019.35 for iteration 12 (LM parameter = 3.42E-17)
- Note: lowest chi-sq **NOT** obtained by standard (*) LM parameter value

Conclusions

- The ORNL Optimization Code has been parallelized with little modification of underlying FORTRAN code
- Scalar and superscalar speed enhancements have been achieved
- Future considerations
 - add parallel global search step