Parallel Finite Element Programming in C++ using OpenMP

EWOMP '03 OMP*lab* Project Proposal

Jörg Multhoff, Mathematical Models in Material Science, RWTH Aachen
multhoff@mmw.rwth-aachen.de

An existing experimental object-oriented C++ finite element code for solid mechanics analysis has been ported to the SunFire 6800 and SunFire 15k platforms using the guidec++ compiler. The code is based on element-by-element techniques both in forming the system of equations and in the solution process using an PCG solver. The code has been parallelized using OpenMP directives and exploiting the loop-level parallelism of the element-by-element approach. The parallelization shows good results for up to 8 processors resulting in about 80 % overall parallel efficiency. However, higher number of processors are not effective with the present code on the Sun platform for unknown reasons. The purpose of this project is to analyse the runtime behaviour of the code for higher number of processors on all of the available shared memory platforms using the latest compilers, tools and - most importantly - help of the available experts.

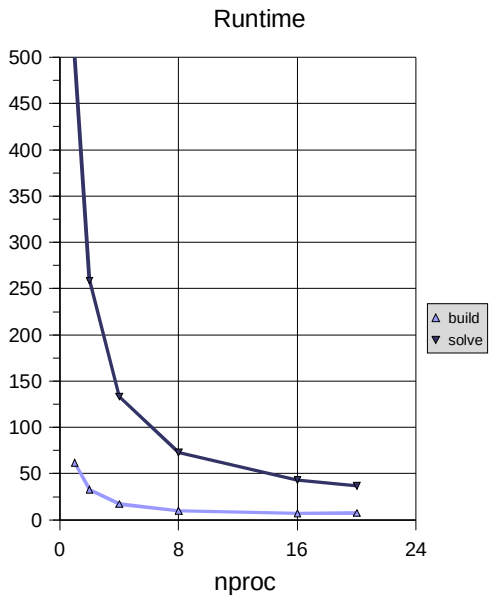
The project will involve the following steps:

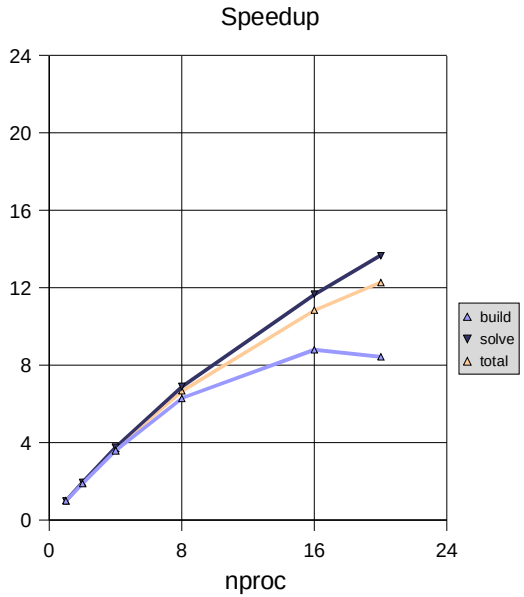
1. Porting the code to the IBM Regatta and Fujitsu Primepower systems. The only issue should be the selection of appropriate compiler switches, if possible with the help of the respective compiler expert.
2. Running an existing benchmark problem on the 3 available systems using different problem sizes and number of processors and using the available runtime analysis tools.
3. Evaluation of the runtime analysis results using the tool developer's experiences.
4. If possible/necessary, make changes to the code and verify the results.

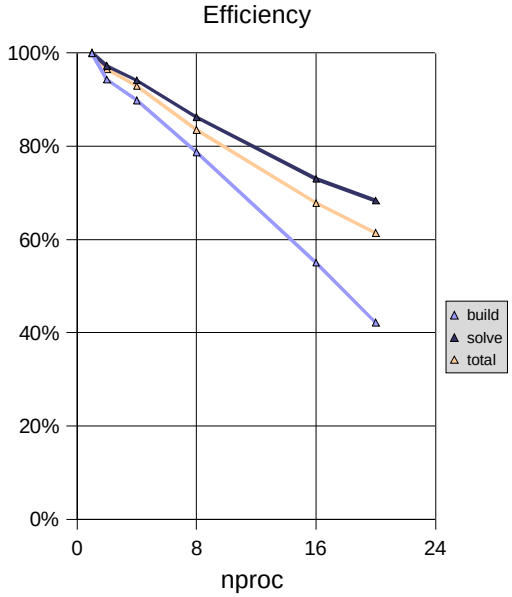
The project should uncover the nature of the code's present scalability problems on the Sun platform and should show if they are persistent when moving to the IBM and Fujitsu platforms. The tool developers have the chance to see the interaction of C++, OpenMP and there respective tools when dealing with the present real-world code.

The attached pages show two typical results for the code's parallel performance .

nproc	Runtime			Speedup			Efficiency		
	build	solve	total	build	solve	total	build	solve	total
1	61.7	502.6	09:27.97	1	1	1	100.0%	100.0%	100.0%
2	32.7	258.4	04:54.26	1.89	1.94	1.93	94.3%	97.2%	96.5%
4	17.2	133.5	02:32.83	3.59	3.77	3.72	89.8%	94.1%	92.9%
8	9.8	72.8	01:25.00	6.3	6.9	6.68	78.7%	86.3%	83.5%
16	7.0	43.0	00:52.35	8.8	11.68	10.85	55.0%	73.0%	67.8%
20	7.3	36.8	00:46.23	8.44	13.67	12.29	42.2%	68.3%	61.4%







nproc	Runtime			Speedup			Efficiency		
	build	solve	total	build	solve	total	build	solve	total
1	18.5	217.5	03:59.44	1	1	1	100.0%	100.0%	100.0%
2	13.5	110.0	02:06.56	1.37	1.98	1.89	68.5%	98.9%	94.6%
4	9.6	58.5	01:11.21	1.92	3.72	3.36	48.1%	93.0%	84.1%
8	11.4	32.3	00:46.44	1.62	6.73	5.16	20.3%	84.1%	64.4%
16	20.1	18.5	00:41.33	0.92	11.78	5.79	5.7%	73.6%	36.2%
20	24.5	15.9	00:43.36	0.75	13.7	5.52	3.8%	68.5%	27.6%