ICPC Technical Report 2001a
Performance models for linear solvers within the component framework

Nathalie Furmento, Anthony Mayer, Stephen McGough,
Steven Newhouse, and John Darlington

Imperial College of Science, Technology and Medicine, Department of Computing,
Huxley Building, 180 Queen's Gate, London SW7 2BZ, UK
icpc-sw@doc.ic.ac.uk
http://www-icpc.doc.ic.ac.uk/components/

Abstract. This technical report details the performance models used for determining the most optimal selection of components for our system. The techniques for generating these performance models is also described.

1 Introduction

In order to determine the best selection of component implementations, based on execution time, cost or some other criteria, it is essential to be able to predict with some level of accuracy the requirements of each implementation. Computation of resource requirements such as memory usage can normally be determined accurately from inspection of the source code. However, determining the execution time for a particular implementation may not be as straight forward.

In this report a number of performance models are presented for the generation of sets of linear equations and their solution via different techniques. The technique used for generating these models is also outlined.

Tables 1 and 2 indicate the source and solver implementations that are available for this report.

System	Processor	Processors	Language
PC (Linux)	AMD 900Mhz	1	Java
Alpha (Atlas)	Alpha 667Mhz	1,4,9,16	ScaLAPACK

Table 1. Available source implementations

System	Processor	Processors	Language	Solution
PC (Linux)	AMD 900Mhz	1	Java	LU
				BCG
			C	LU
				BCG
			LAPACK	LU
Alpha (Atlas)	Alpha 667Mhz	1,4,9,16	ScaLAPACK	LU
				BCG
AP3000	UltraSPARC 300Mhz	4,9,16	ScaLAPACK	LU
				BCG

Table 2. Available solver implementations

2 Generating Performance Models

In all cases the models are based on actual performance results. The technique for generating these modles is outlined below, illustrated by the 9 processor linear equation generator on the Atlas. All other performance models were generated in a similar manner.

A simple Java wrapper is written around the implementation. This allows for the implementation to be executed and timings to be recorded. A set of these execution timings can be seen in figure 1 below.

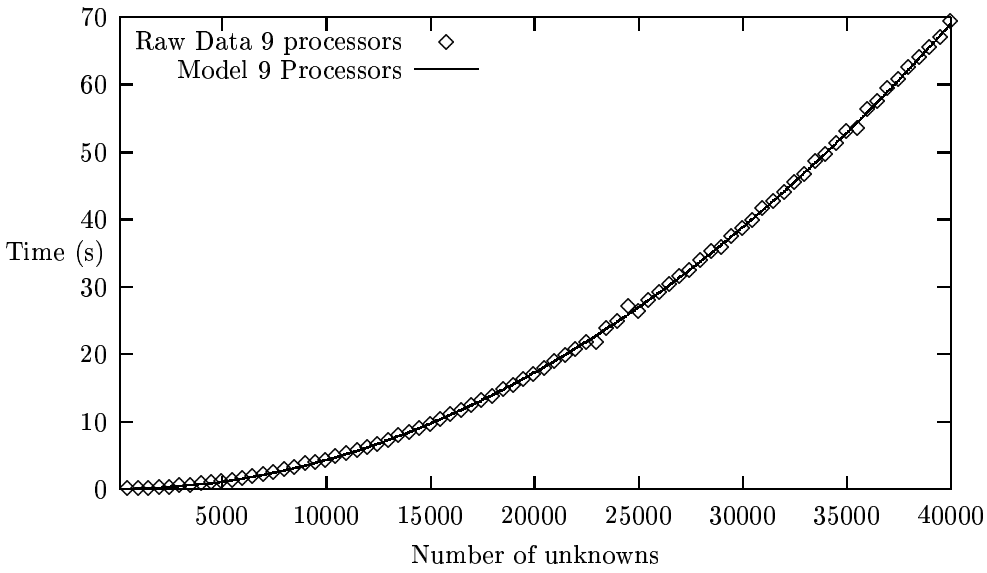

Fig. 1. Raw results for a 3×3 grid generating sets of linear equations.

A curve is then generated that approximates the initial data set using the least squares approach. Polynomials of varying degrees (1-10) are tried in order to determine which gives the best fit. Often with these curves, a different polynomial is required for different ranges of unknowns. The initial polynomial is checked for points at which it fits the original data well and points where it is most inaccurate. Polynomials are then fitted to each of these intervals, again trying to determine the best degree. Each sub-interval curve is then compared along with the adjacent curves and the initial data to determine the best point to switch between adjacent curves. In the case of the nine processor generator this gives two curves:

if $(size > -1.0 \;\&\&\; size <= 4101.0)$
$time = 1.2551442197446745^{-8} + 2.7144007296198618^{-5} * size +$
$4.0451867125361384^{-8} * size * size$
else
$time = 4.4399451058772966^{-17} + 1.3439520756667027^{-12} * size +$
$4.313062339573421^{-8} * size * size$

In the case of the Biconjugate Gradient method two curves are generated for each implementation. The first represents the startup time required by the solver, and the second is the time required for each pass through the iteration loop. These can be combined to give the total predicted execution time for the implementation:

$$time = startup + i * iterationTime, \tag{1}$$

where i is the number of iterations to be performed.

3 Performance Models

Figure 2 shows the expected performance models for the generation of linear equations. Figure 3 illustrate the performance models for generating the linear equations in parallel for the Atlas cluster. It can be seen from these graphs that there are optimal implementations to select for different numbers of unknowns.

Figures 4, 5 and 6 illustrate the performance models for solving the systems of linear equations on the sequential Linux workstation and the clusters of Sun and Alpha processors.

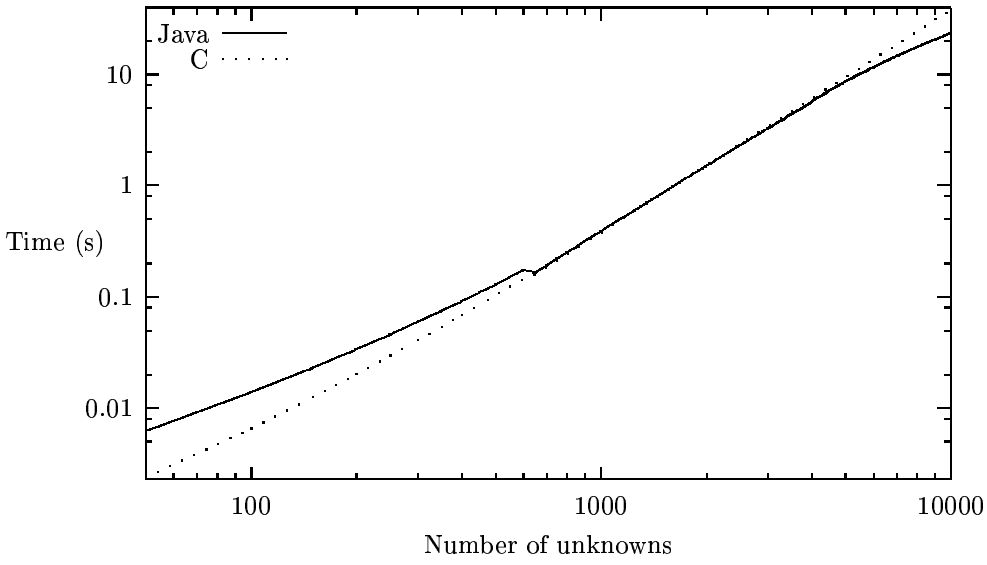

Fig. 2. Performance model for single processor Linux generation of linear equations

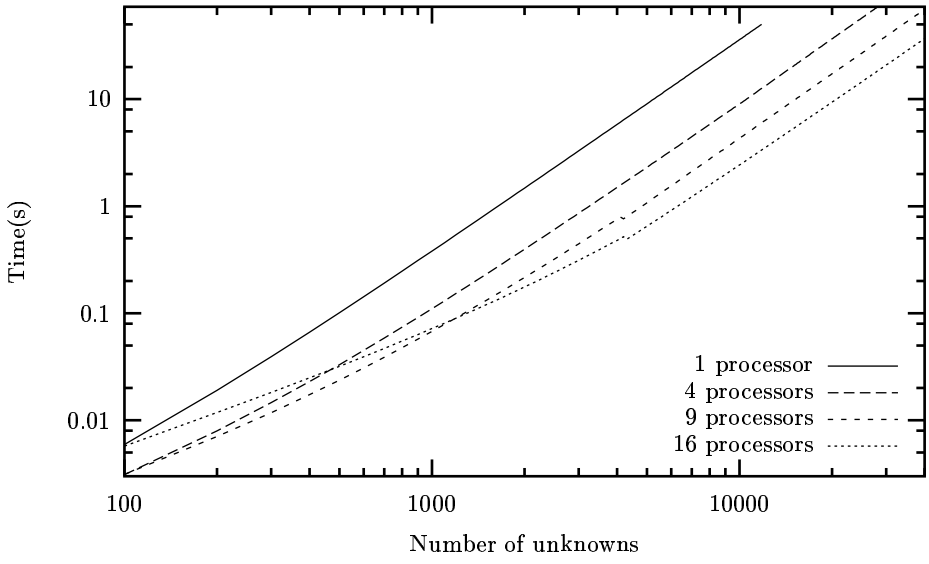

Fig. 3. Performance model for parallel generation of linear equations on a cluster of Alpha processors

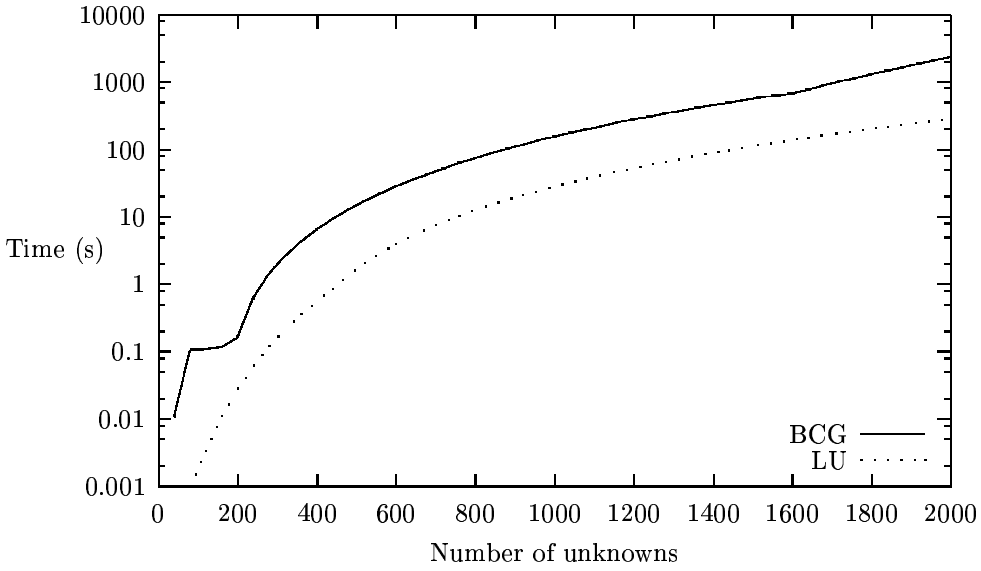

Fig. 4. Performance model for single processor Linux solvers

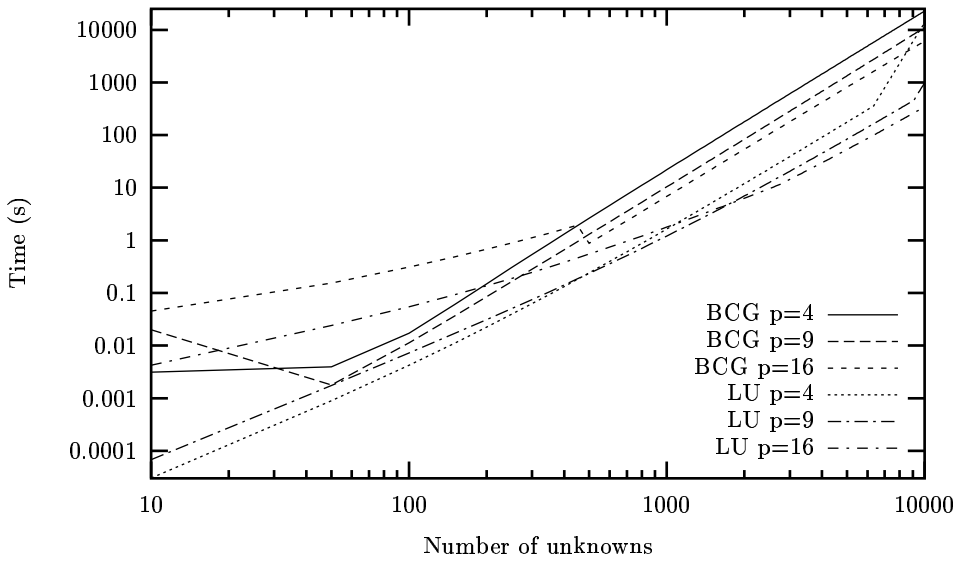

Fig. 5. Performance model for parallel solvers on a cluster of Sun processors

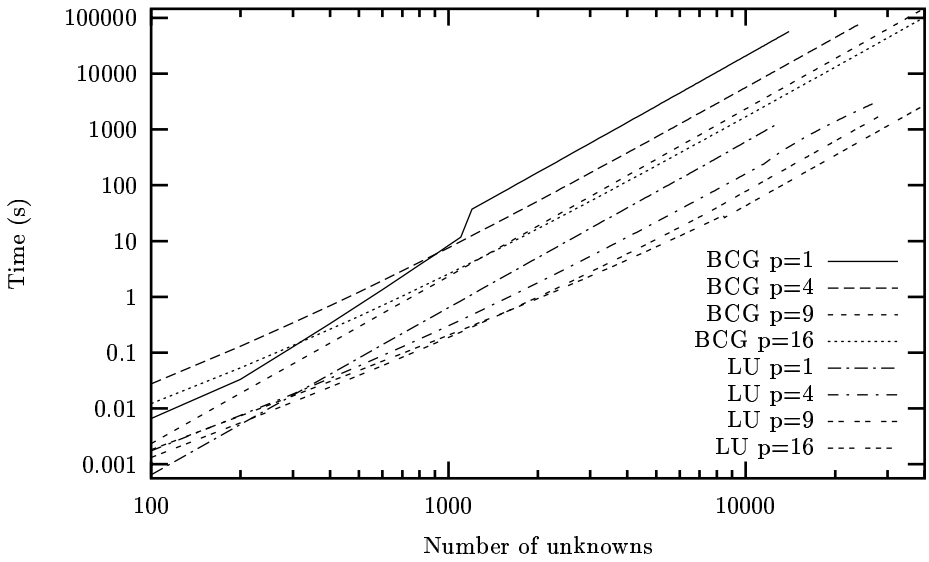

Fig. 6. Performance model for parallel solvers on a cluster of Alpha processors

The following models utilise knowledge of the linear system's diagonal dominance:

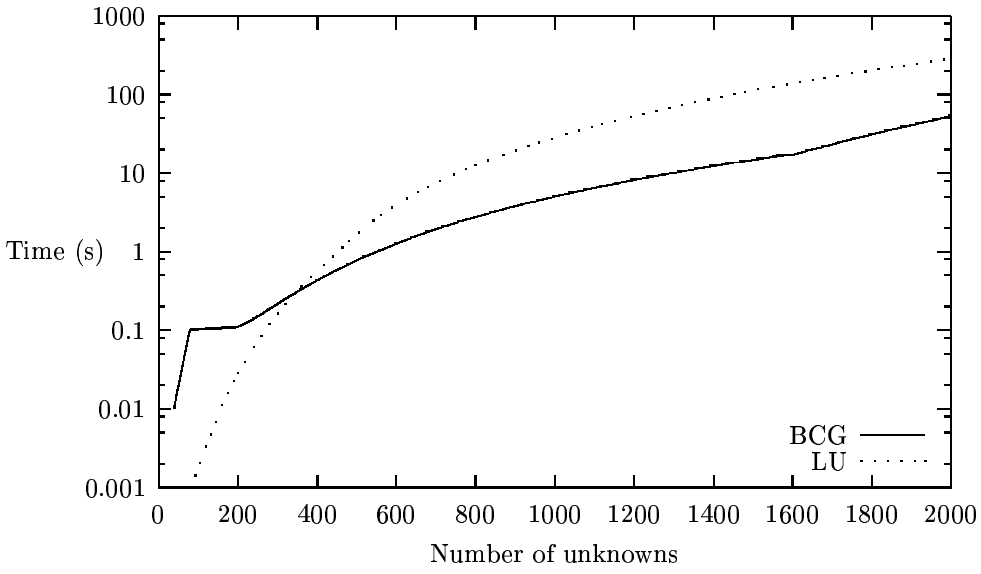

Fig. 7. Performance model for single processor Linux solver $i = \sqrt{n}$

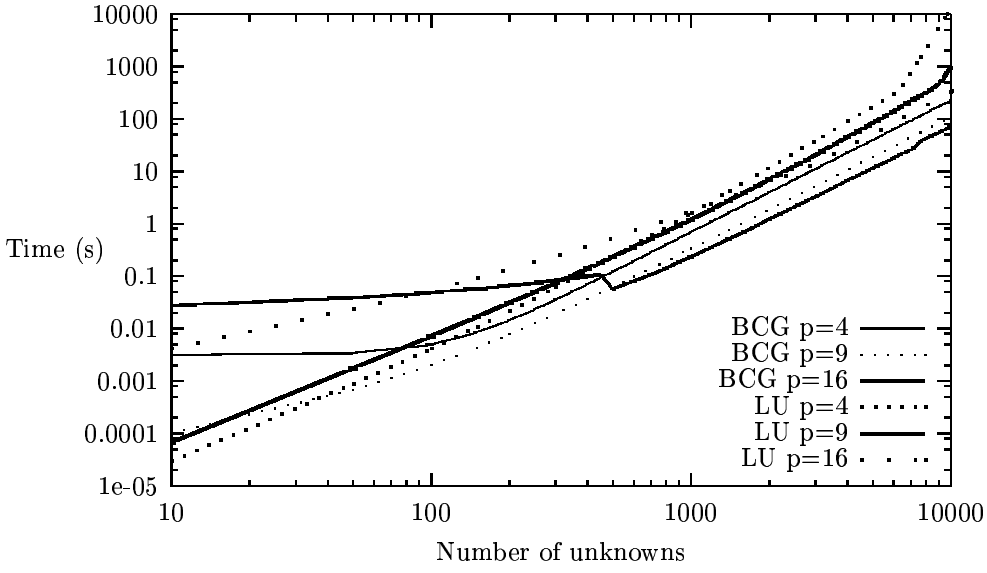

Fig. 8. Performance model for parallel solver $i = \sqrt{n}$ on a cluster of Sun processors

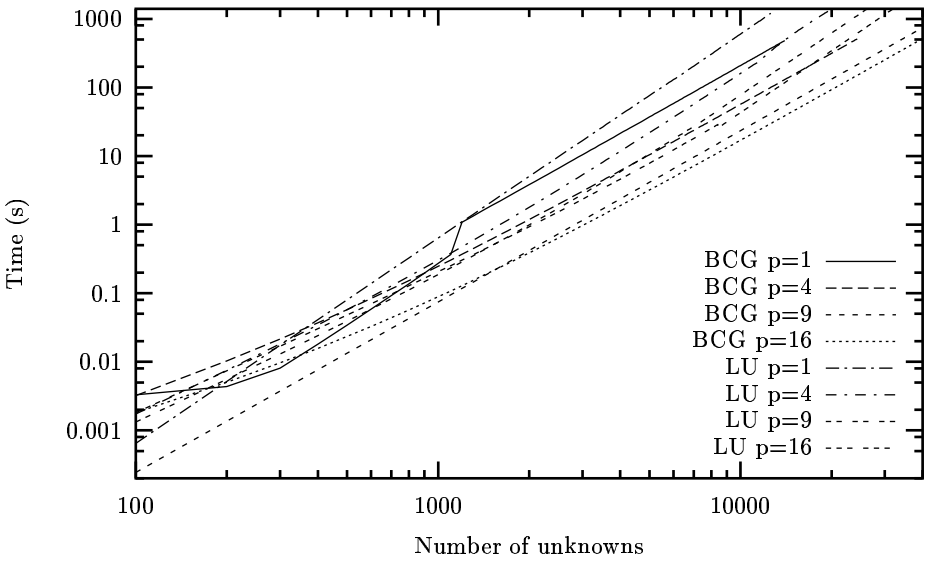

Fig. 9. Performance model for parallel solver $i = \sqrt{n}$ on a cluster of Alpha processors

Figure 7 illustrates the expected execution times for the solvers on the Linux box when the digonal dominance is chosen such that the numnber of iterations required for the BCG solver is approximatly equal to the square root of the number of unknowns. Likewise figures 8 and 9 illustrate the expected execution times on the AP3000 and Atlas clusters, under the same diagonal dominance.

It can be seen from Figures 8 and 9 that where there are a large number of unknowns in a system deployed upon a small number of processors the predicted execution time may become very large. This is due to the limited memory available to each processor. Such characteristics may be included within the CXML component descriptions.