

OpenMP Microbenchmarks Version 2.0

Fiona J. L. Reid and J. Mark Bull

EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, U.K.

f.reid@epcc.ed.ac.uk or m.bull@epcc.ed.ac.uk

Abstract

Overheads due to synchronisation, loop scheduling and array operations are an important factor in determining the performance of shared memory parallel programs. We present an updated set of benchmarks to measure these classes of overhead for the language constructs used in OpenMP. Results are presented for a Sun Fire 15K, an IBM p690+ and an SGI Altix, each with its own implementation of OpenMP. Significant differences between the implementations are observed, which suggest possible means of improving future performance.

1. Introduction

Synchronisation, loop scheduling and array operations can all be significant sources of overhead in shared memory parallel programs. In OpenMP ([4], [3]), the cost of these operations is dependent on their implementation in the OpenMP runtime library. In this paper we describe techniques for measuring the overheads associated with synchronisation directives, scheduling directives and array operations. We present results for OpenMP Fortran 90 implementations on a Sun Fire 15K, an IBM p690+ and an SGI Altix.

We present version 2.0 of the OpenMP microbenchmark codes which represent a significant update to version 1.0 released in 1999 [1] and the subsequent update described in [2]. The extended microbenchmark suite presented here allows the measurement of OpenMP 2.0 features and also includes a new benchmark which measures the overheads associated with array operations. The codes are available in C and Fortran 90 and can be downloaded from www.epcc.ed.ac.uk/research/openmpbench

The changes made to version 2.0 of the OpenMP microbenchmark codes are summarised below:

- A new benchmark measuring the overhead associated with various clauses when applied to arrays has been

added. This benchmark measures the overhead of the `PARALLEL` directive with the `PRIVATE`, `FIRSTPRIVATE`, `COPYIN` and `REDUCTION` clauses and of the `SINGLE` directive with the `COPYPRIVATE` clause.

- Two new synchronisation benchmarks have been added to measure the overheads of the `WORKSHARE` and `PARALLEL WORKSHARE` directives.
- The Fortran codes have been re-written to use Fortran 90 syntax and free format; common blocks and include files have been replaced by modules.
- The repeat counts have been reduced where possible to speedup runtimes.
- `#ifdef` statements have been added to allow for OpenMP 1.0/2.0 compatibility
- The timing routines now use `OMP_GET_WTIME()` if OpenMP 2.0 is available. If OpenMP 2.0 is not available, `system_clock()` is used for the Fortran codes and `get_time_of_day()` is used for the C codes.

2. Methodology

To measure the overhead of OpenMP directives, the technique used is to compare the time taken for a section of code executed sequentially with the time taken for the same code executed in parallel enclosed in a given directive. A full description of this method is given in [1] and [2].

To obtain accurate results, careful choice of clock routine is necessary. Second differences of raw clock values are used to compute the overheads; the runtime for each measurement cannot be too large, since the measurements need to be repeated many times for statistical stability. Therefore, it is essential to ensure that the values returned by the clock routine are not only sufficiently accurate (typically to the nearest microsecond) but also sufficiently precise (preferably to 64-bit). All the results presented in this paper use the OpenMP runtime library routine `OMP_GET_WTIME()`

for the timing calculations. This routine returns 64-bit floating point values and is accurate to the nearest microsecond on the three systems measured.

To obtain statistically meaningful results, each overhead measurement is repeated 20 times within a single run and 20 different runs are performed. This means that each measurement is obtained from the average of 400 individual measurements. The reason for this is that a much larger variability is observed between different runs than between different measurements within the same run. The cause of this is unknown but it may be due to the physical memory locations of synchronisation variables altering from run to run. Although every effort is made to ensure we have exclusive access to a given machine/processors, this is not always achievable. Background processes (e.g. OS related) may also be contributing the variability observed between runs.

Within each run we compute the mean and standard deviation σ of all 20 measurements. Since we do not have exclusive access to the hardware platforms, we test for “clean” runs by checking for large values of σ and also for cases where there are many outliers (values more than 3σ above the mean).

3. Results

The benchmarks were run on the following systems: a Sun Fire 15K with 52 900MHz Ultrasparc III processors using Forte Developer, Fortran 95 version 7.0, an IBM p690+ with 32 1700MHz POWER4+ processors using XL Fortran for AIX, version 8.1.1 and an SGI Altix with 256 1300MHz Itanium 2 processors using the Intel Fortran Compiler for Linux, version 7.1. Both the Sun Fire 15K and IBM p690+ had OpenMP 2.0 compatible compilers installed. The SGI Altix did not have an OpenMP 2.0 compiler installed and therefore only results for OpenMP 1.0 directives are presented for it.

The scheduling and array benchmarks were run using 8 processors. The synchronisation benchmarks and the array benchmark for a fixed array size were run for various numbers of processors from one up to the maximum possible for the particular machine. The maximum number of processors available to a shared memory code were respectively 48, 32 and 62 for the Sun Fire 15K, IBM p690+ and SGI Altix.

Figures 1 to 3 show the measured overheads against the number of processors for the implied barrier directives on the three systems. Measurements for the WORKSHARE and PARALLEL WORKSHARE directives could not be obtained for the SGI Altix (Figure 3) as they are OpenMP 2.0 specific features.

Figures 4 to 6 show the overheads of the various clauses with array arguments against array size on the three systems. Measurements for the COPYPRIVATE and REDUCTION clauses could not be obtained on the SGI Altix as they are both OpenMP 2.0 features.

Figures 7 to 9 show the measured overheads against chunk size for the different loop schedules on the three systems.

4. Analysis

Figure 1 shows that the BARRIER, DO, SINGLE and WORKSHARE directives have similar performance on the Sun Fire 15K, suggesting that nearly all the cost of the DO, SINGLE and WORKSHARE directives is in the implied barrier. All four directives appear to scale well with increasing numbers of processors. The PARALLEL, PARALLEL DO and PARALLEL WORKSHARE directives take between 2 and 9 times as long as the BARRIER directive, with cost increasing steadily with the number of processors. A slight increase in cost is observed at around 44 processors that may be spurious: high standard deviations were observed for these results. The addition of a REDUCTION clause to the PARALLEL directive significantly increases its cost and makes it less scalable, especially for more than eight processors.

Figure 2 shows that the BARRIER, DO, SINGLE and WORKSHARE directives behave in a similar way on the IBM p690+ as on the Sun Fire 15K, except that the number of clock cycles is up to 12 times higher on the IBM p690+. Even allowing for the difference in clock speed between the machines these differences are large. The PARALLEL and PARALLEL DO directives are more costly (approx 30% greater for 24 processors) than the BARRIER directive. Unlike the Sun Fire 15K, the PARALLEL + REDUCTION directive costs approximately the same as the PARALLEL directive. This means that the cost associated with the REDUCTION is nearly all attributed to the cost of the PARALLEL directive. The PARALLEL WORKSHARE directive on the IBM p690+ scales very poorly, and is more than double the cost of using PARALLEL. All the overheads on the IBM p690+ show a distinct change in gradient at 8 processors. This is not entirely unexpected as each 32 processor frame of the IBM p690+ comprises of four eight processor Multi-Chip Modules (MCM). It appears that communications within an MCM are significantly less costly than those between MCM's where the communications need to go via the bus interconnect.

On the SGI Altix (Figure 3) the BARRIER, DO and PARALLEL DO directives have similar performance

and cost. As with the Sun Fire 15K and IBM p690+ the PARALLEL directive is more costly (up to a factor of two) than the BARRIER directive. The main difference is that the REDUCTION and SINGLE directives have almost the same overheads associated with them. The cost of the SINGLE directive is always greatest on the SGI Altix, e.g. for 32 processors the Altix is over 10 times slower than the Sun Fire 15K and nearly twice as slow as the IBM p690+.

Figure 4 shows the overheads of the various clauses with array arguments for the Sun Fire 15K. All the overheads except PRIVATE show a steady, linear increase with array size. The cost of the PRIVATE clause is approximately the same as a PARALLEL directive and does not vary with array size. The overhead of the COPYPRIVATE clause exhibits rather erratic behaviour for small (<729) array sizes, beyond which a steady increase is observed. The reason for this erratic behaviour is unknown. The REDUCTION clause is always the most expensive and costs more than two million clock cycles for the array size of 59049.

On the IBM p690+ (Figure 5) the overhead associated with the PRIVATE clause behaves similarly to the Sun Fire 15K, except that it is 50% less expensive on the IBM p690+. The COPYPRIVATE clause is significantly cheaper on the IBM p690+ and does not show the erratic behaviour observed on the Sun Fire 15K for small array sizes. The FIRSTPRIVATE and COPYIN clauses have nearly identical costs that increase linearly with array size. A sharp change in gradient is observed at array size 19683 suggesting that we may have encountered some architecture or implementation related limit. The overhead of the REDUCTION clause increases linearly with array size until 6561 and then levels out for larger array sizes.

Figure 6 shows the corresponding results for array clauses on the SGI Altix. The PRIVATE clause shows somewhat erratic jumps at array sizes 27, 729 and 2187; again the reason for these jumps is currently unknown. The FIRSTPRIVATE and COPYIN clauses behave in much the same way as the other two systems, except that the number cycles required is less - up to 85 times faster than the IBM p690+ and up to 11 times faster than the Sun Fire 15K.

Figure 7 shows that on the Sun Fire 15K the block cyclic scheduling (STATIC,N) costs the same as a block schedule (STATIC) for chunk sizes greater than eight. For small chunk sizes (from 1 to 8) the overhead decreases rapidly with increasing chunk size toward the STATIC value. DYNAMIC scheduling is several orders of magnitude more expensive and shows a logarithmic

decrease with chunk size. GUIDED scheduling is over ten times as expensive as the STATIC,N schedule.

On the IBM p690+ (Figure 8) the overhead of the block cyclic schedule does not converge to that of the block schedule until the chunk size exceeds 32. Unlike the Sun Fire 15K, the overhead of the DYNAMIC schedule converges to that of the block cyclic schedule for chunk sizes of 64 or greater. The GUIDED schedule behaves in a similar way to the Sun Fire 15K. The DYNAMIC and GUIDED schedules on the IBM p690+ are up to three times cheaper than on the Sun Fire 15K.

Figure 9 shows that on the SGI Altix, the overheads of the block cyclic schedule and block schedule are approximately the same for all chunk sizes. Curiously, and unlike the other two systems, the GUIDED schedule is always more expensive than the DYNAMIC schedule and shows no sign of converging toward the STATIC values. Comparison with the other two systems shows that the overhead of the GUIDED schedule on the SGI Altix is more than twice that of the IBM p690+ and four times that of the Sun Fire 15K. The DYNAMIC schedule behaves in a similar manner to the other two systems and is consistently smaller, particularly for small (<32) chunk sizes.

5. Conclusions

This paper has presented an updated set of benchmarks for measuring the overheads of synchronisation, loop scheduling and array operations in OpenMP 2.0. Particular emphasis has been placed on obtaining statistically meaningful measurements. The benchmarks have been run on three distinct shared memory platforms. Differences are observed both between the individual directives and between the same directive examined on each system. Some potential areas for optimisation have also been highlighted.

6. References

- [1] J.M. Bull, "Measuring Synchronisation and Scheduling Overheads in OpenMP", *Proceedings of the First European Workshop on OpenMP*, Lund, Sweden, 1999, pp. 99-105.
- [2] J.M. Bull and D. O'Neill, "A Microbenchmark Suite for OpenMP 2.0", *Proceedings of the Third European Workshop on OpenMP (EWOMP'01)*, Barcelona, Spain, 2001.
- [3] OpenMP Architecture Review Board, "OpenMP C and C++ Application Program Interface", Version 2.0, March 2002.
- [4] OpenMP Architecture Review Board, "OpenMP Fortran Application Program Interface", Version 2.0, November 2000.

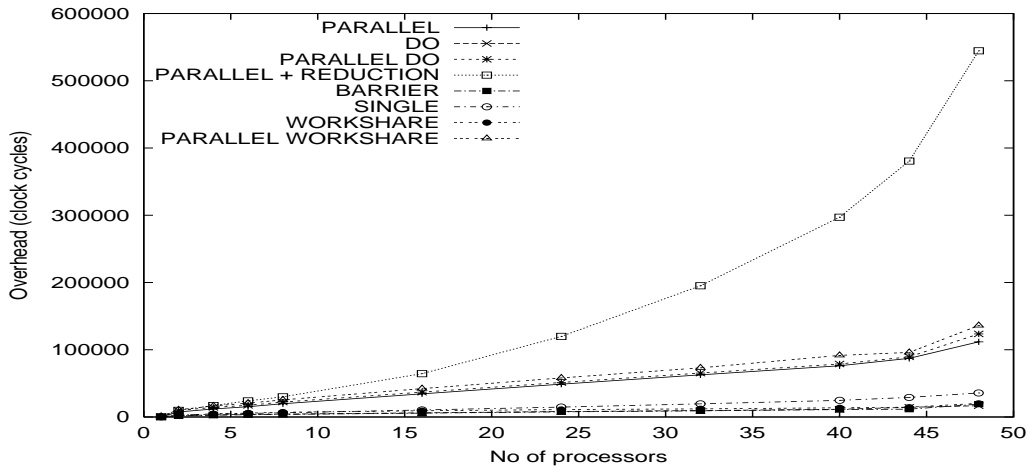


Figure 1: Synchronisation overheads on Sun Fire 15K with Forte Developer 7.0 compiler

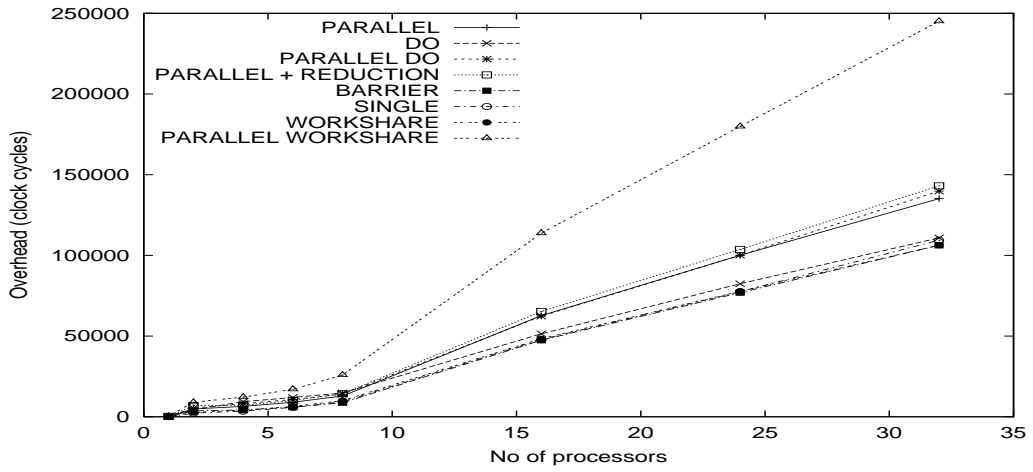


Figure 2: Synchronisation overheads on IBM p690+ with XL Fortran for AIX version 8.1.1 compiler

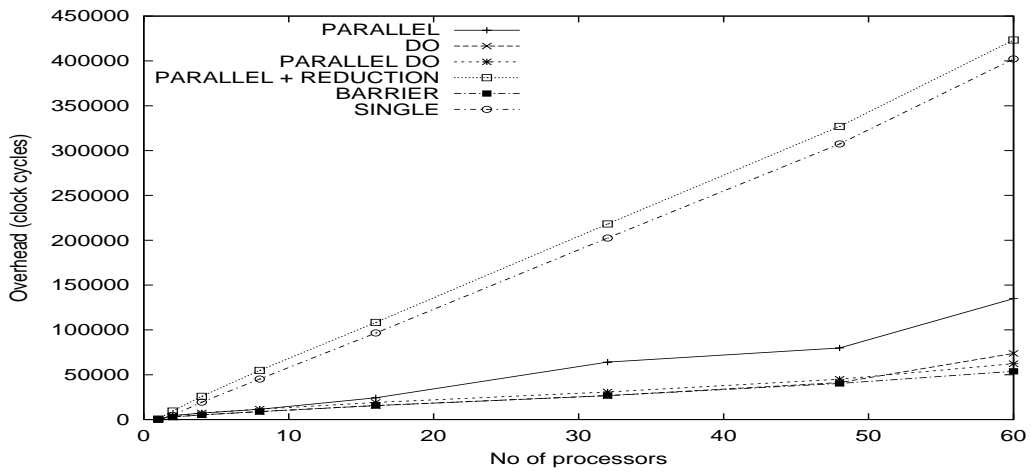


Figure 3: Synchronisation overheads on SGI Altix with Intel Fortran Compiler version 7.1

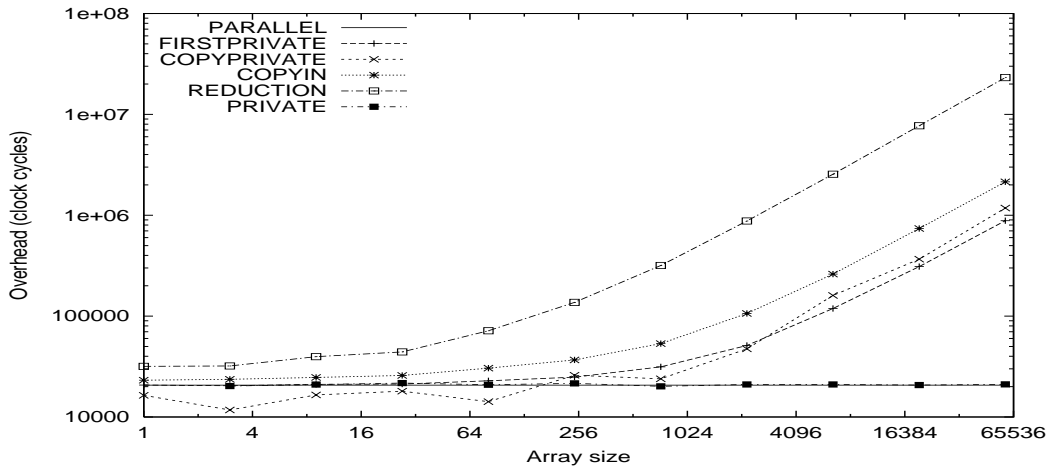


Figure 4: Array overheads on Sun Fire 15K with Forte Developer 7.0 compiler

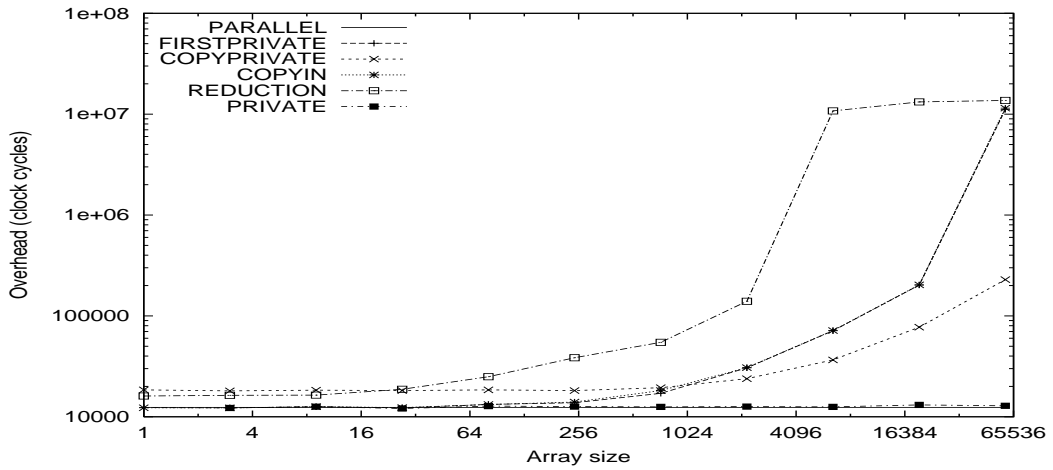


Figure 5: Array overheads on IBM p690+ with XL Fortran for AIX version 8.1.1 compiler

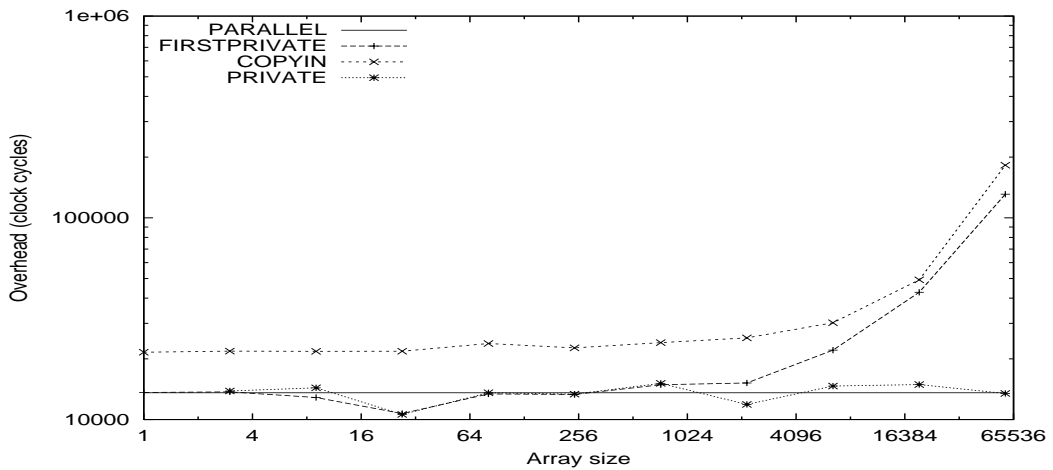


Figure 6: Array overheads on SGI Altix with Intel Fortran Compiler version 7.1

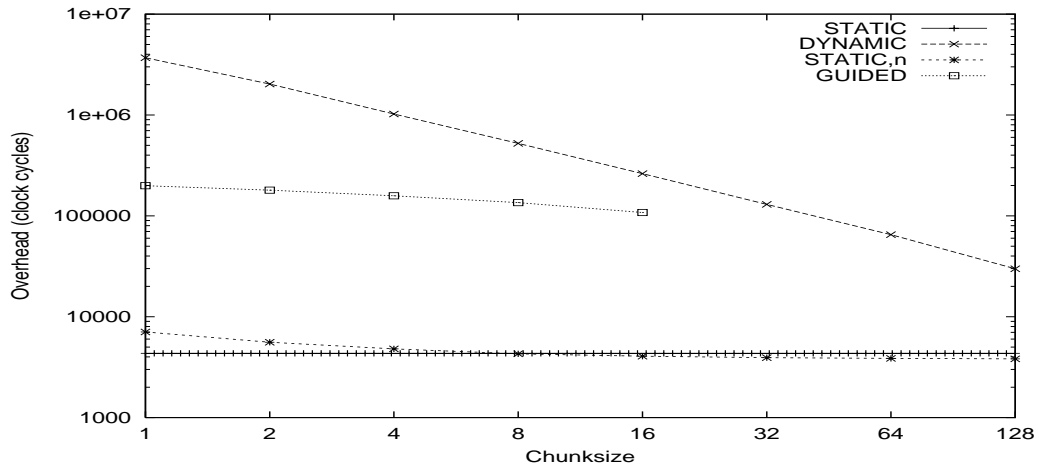


Figure 7: Scheduling overheads on Sun Fire 15K with Forte Developer 7.0 compiler

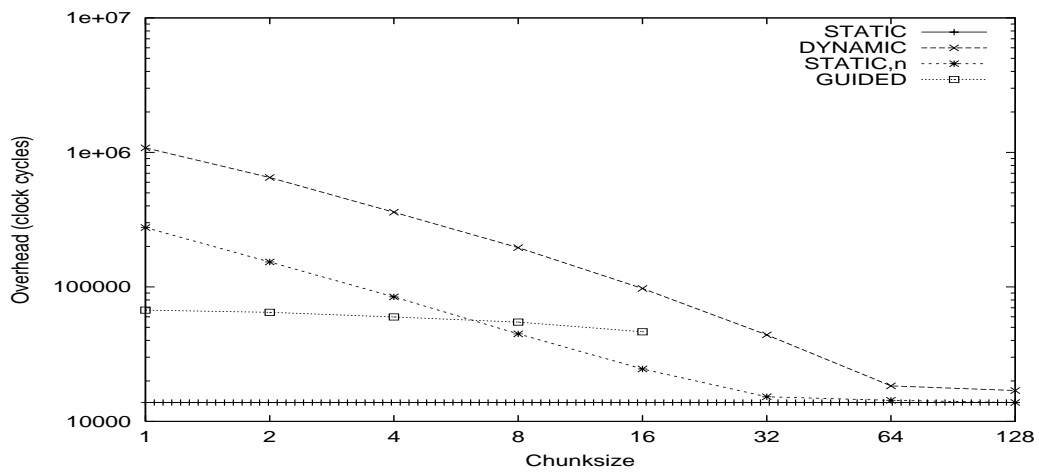


Figure 8: Scheduling overheads on IBM p690+ with XL Fortran for AIX version 8.1.1 compiler

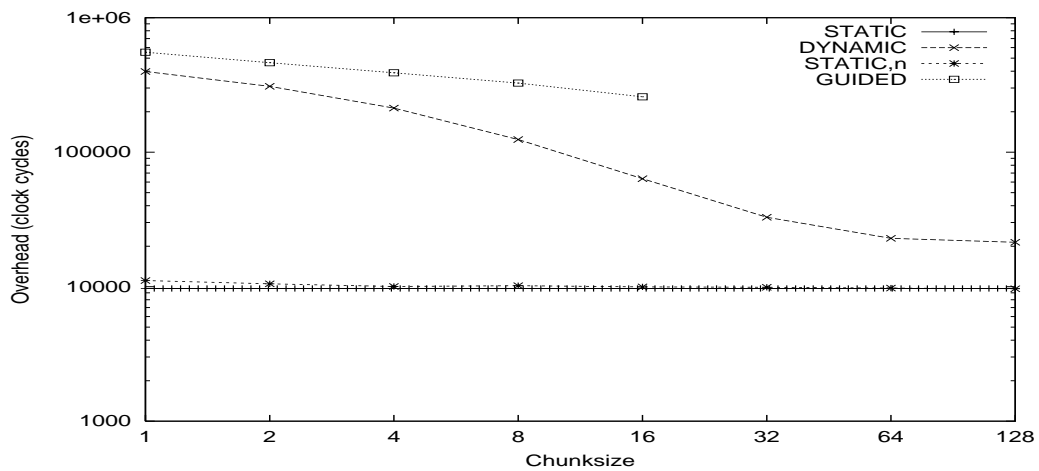


Figure 9: Scheduling overheads on SGI Altix with Intel Fortran Compiler version 7.1