Multicore in production: advantages and limits of the multi-process approach in the ATLAS experiment

S Binet[1], P Calafiura[2], M K Jha[3], W Lavrijsen[2], C Leggett[2], D Lesny[4], H Severini[5], D Smith[6], S Snyder[7], M Tatarkhanov[2], V Tsulaia[2], P VanGemmeren[8] and A Washbrook[9]

[1] Laboratoire de l'Accélérateur Linéaire/IN2P3, 91898 Orsay Cédex, France
[2] Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA
[3] INFN, CNAF & Bologna, Bologna 40127, Italy
[4] University of Illinois at Urbana-Champaign, 1110 W Green St, Urbana, IL 61801, USA
[5] University of Oklahoma, 440 W. Brooks Street, Norman, OK 73019, USA
[6] SLAC National Accelerator Laboratory, 2575 Sand Hill Rd, Menlo Park, CA 94025, USA
[7] Brookhaven National Laboratory, P.O. Box 5000, Upton, NY 11973, USA
[8] Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA
[9] SUPA, School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK

E-mail: vtsulaia@lbl.gov

Abstract. The shared memory architecture of multicore CPUs provides HEP developers with the opportunity to reduce the memory footprint of their applications by sharing memory pages between the cores in a processor. ATLAS pioneered the multi-process approach to parallelize HEP applications. Using Linux fork() and the Copy On Write mechanism we implemented a simple event task farm, which allowed us to achieve sharing of almost 80% of memory pages among event worker processes for certain types of reconstruction jobs with negligible CPU overhead. By leaving the task of managing shared memory pages to the operating system, we have been able to parallelize large reconstruction and simulation applications originally written to be run in a single thread of execution with little to no change to the application code. The process of validating AthenaMP for production took ten months of concentrated effort and is expected to continue for several more months. Besides validating the software itself, an important and time-consuming aspect of running multicore applications in production was to configure the ATLAS distributed production system to handle multicore jobs. This entailed defining multicore batch queues, where the unit resource is not a core, but a whole computing node; monitoring the output of many event workers; and adapting the job definition layer to handle computing resources with different event throughputs. We will present scalability and memory usage studies, based on data gathered both on dedicated hardware and at the CERN Computer Center.

1. Introduction

Memory is a scarce resource for typical ATLAS reconstruction jobs and by running many individual jobs simultaneously we are likely to hit hardware memory limits on the production machines. Thus, in order to optimally exploit all available CPU cores on a given node, we need to have a parallel solution which allows us to share resources between processes or threads. The ATLAS experiment pioneered the process based parallelism in HEP applications. We have implemented a simple event task farm, which relies on the Linux kernel Copy On Write (COW) mechanism. The basic strategy of this approach is to go through the initialization stage of a job in a single master process, allocate and initialize as much of the memory that will be used by the job, then fork many worker processes and let them process their share of events of the job. Workers share a large fraction of process memory between each other and such memory sharing is solely provided by the COW mechanism. We have been able to run in parallel large reconstruction and simulation applications originally written to be run in a single thread of execution with little to no change to the application code.

Besides validating the software itself, an important and time-consuming aspect of running multicore applications in production was to configure the ATLAS distributed production system to handle multicore jobs. This entailed defining multicore batch queues, where the unit resource is not a core, but a whole computing node; monitoring the output of many event workers; and adapting the job definition layer to handle computing resources with very different event throughput (depending on the number of cores used).

This paper describes design and implementation concepts of AthenaMP [1] - the Multi-Process version of the ATLAS common reconstruction and analysis framework *Athena*. We present performance, scalability and memory usage studies based on data gathered both on dedicated hardware and at the CERN Computer Center.

In section 2 we go through details of AthenaMP implementation with the emphasis on minimized user code changes. Section 3 is devoted to running AthenaMP jobs on the Grid. Section 4 discusses AthenaMP validation process and also describes a series of large scale AthenaMP reconstruction tests performed over Summer 2011 at the CERN Computer Center ("Tier-0" of the LHC Computing Grid). In section 5 we present performance figures comparing AthenaMP reconstruction to N individual Athena processes run in parallel on the same machine. Finally, section 6 presents several ways to further improve AthenaMP performance, in particular by increasing reconstruction job size and by implementing specialized worker processes, which will be in charge of the event I/O operations.

2. AthenaMP

2.1. Sharing memory between processes using Copy On Write

It is trivial to achieve process-based parallelism on a multicore host. The simplest scenario, which *a priori* requires no code changes, is just to run N instances of the same application simultaneously. However, the obvious problem with this approach is the suboptimal usage of system resources. Nothing apart from dynamic shared objects in memory and files opened in read only mode is shared between these application instances. Such trivial multi-process architecture leads to increased memory requirements, which rapidly scale with number of individual processes.

One possible way to solve the memory duplication problem is to allocate and initialize as much memory as possible during the initialization phase of a single parent process and then fork() N child processes (workers). Upon invocation fork() clones the parent process including its entire address space. This may sound as counter-productive for the problem at hand, however modern operating systems use Copy On Write mechanism as an efficient optimization for such cases. The child is not given a private copy of the process memory, instead parent and child processes share physical memory up to the point when one of them writes to it. At that point, the affected memory page will be copied and become a 'property' of the writer process. Obviously the optimal strategy for such architecture is to fork() as late as possible in order to achieve maximal memory sharing between processes. The effect of such late forking can be quite dramatic and this is demonstrated by the performance figures in section 4 of this paper.

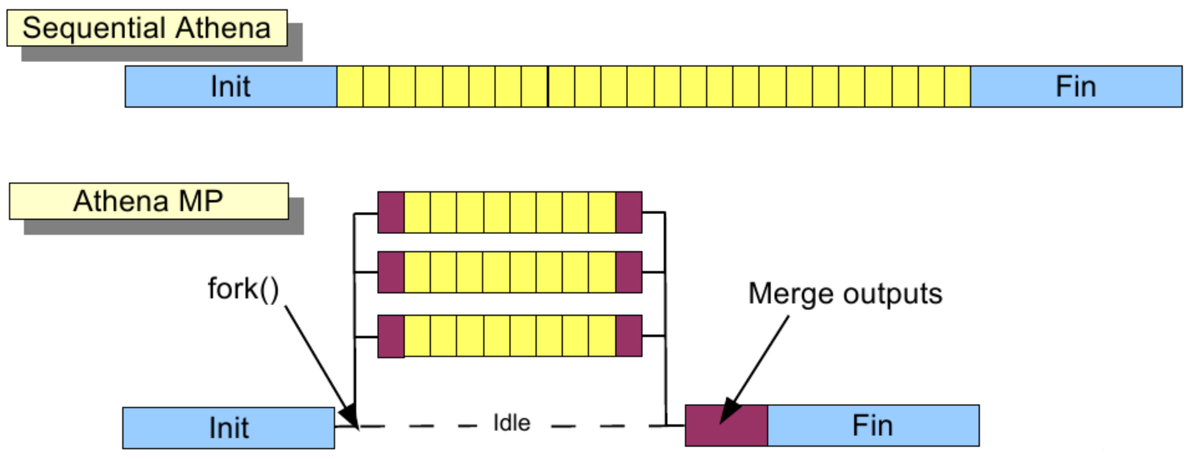

Figure 1. Athena MP versus sequential Athena job.

Another big advantage of the fork-based multi-process approach is that each child process can continue its work independently with little to no communication with either its parent process or other workers. As a result no changes are required in the user code for serializing access to shared data and to critical sections of the code under the assumption that all communications between processes will be handled by the framework itself.

Finally, relying on COW ensures that as much memory as possible is shared and automatically done so by the Linux kernel.

The main disadvantage of this approach is that once a memory page became unshared due to a write, it cannot be shared again later on. This may not be a real issue during event processing stage of the worker processes, assuming that workers don't need to allocate large chunks of memory individually. However, the implications of the memory un-sharing can be quite severe and we have observed huge spikes in overall memory consumption at the finalization stage of worker processes, which introduce considerable performance penalties when we hit machine memory limits. We believe that these memory spikes are caused by simultaneous calling of object destructors by worker processes. However the exact reason is yet to be understood.

2.2. Process management and output file handling

One of the constraints of parallelizing Athena is to avoid client code changes: AthenaMP fulfills this condition by hiding multi-process semantics inside the Athena/Gaudi [2] framework instead of publishing them as a layer atop.

Process steering in AthenaMP is implemented using python multiprocessing module [3]. A pool of forked sub-processes is created just before entering the event loop. The child processes are given a bootstrap function to handle I/O reinitialization (reopen file descriptors) and asynchronously scheduled to process all input events. The events are distributed between workers on a first come first served basis using a shared queue, which provides workers with event sequence numbers. Besides accessing common input files the event workers run independently in separate run directories, they don't communicate to one another and make their own output files. In the current implementation of AthenaMP workers process events one by one. For the future we plan to change this approach and let the workers process events in chunks in order to avoid reading the same portion of the input ROOT file by multiple processes and by this way improve overall performance of AthenaMP jobs.

When there are no more events to process the workers go through finalization stage and exit. Once all of the workers have terminated the control is passed back to the parent process, which then proceeds with merging workers' output files. This is probably the most problematic part of AthenaMP as POOL [4], the object persistency library used by ATLAS, does not support parallel I/O. Thus the parent process has to merge all types of outputs in serial and even after recent switch to the fast merger utility for POOL files, which improved merging performance by one order of magnitude, the output merging remains one of the critical areas which has a negative impact on the overall performance of the Athena MP.

3. AthenaMP on the Grid

In order to run AthenaMP reconstruction jobs on the ATLAS distributed production system it was necessary to define special multicore batch queues, where the unit resource is a whole computing node. As well as the whole-node production queue at CERN there are Tier-1 and Tier-2 Grid sites in the UK (RAL, Edinburgh, Glasgow), US (Illinois and OSCER - University of Oklahoma, OU Supercomputing Center for Education and Research) and Italy (INFN-T1, Naples) that have been configured to accept AthenaMP jobs submitted from the ATLAS production system. These candidate sites host a number of different batch system implementations (SGE, Torque/Maui, LSF and Condor) which are representative of the majority of Grid sites available to ATLAS.

In order for these computing resources to be made available for whole-node jobs it was necessary to apply configuration changes to the batch system scheduler at each participating site. At this time there is no prescriptive model on how to configure a Grid site for whole-node job execution. Rather, the choice of configuration was dependent on the existing resource allocation defined by each site.

AthenaMP use on the Grid poses a set of unique difficulties. In local use of AthenaMP, the user knows what hardware is being used, and is able to set the number of processes to match this hardware. But for jobs run on the Grid, this is not the case. The user will be able to submit jobs, but usually is not able to choose the resources where those jobs will run. So the number of processes in use for the job needs to be defined at run time, on the worker node.

The first method to answer this problem, is in the queue definitions on the Grid resources. The jobs will be assigned to certain sites, and from there run on sets of resources defined in a batch queue. In job management systems, there is added information about the queue, in the form of the number of CPU cores that exist in all resources on this queue. At run time, the job management system will use this information to define an environment variable, with the CPU core information. AthenaMP at run time can use the value of this variable to define the number of processes to use in running.

This creates some problems for the remote site and job management. The remote site usually has the CPU resources defined as one job per CPU core as a default. The queues for AthenaMP running have to be defined as all MP jobs, so there will be a certain number of CPUs per job defined. In practice sites should define the number of cores used per job as the number cores per machine in the queue. When a mixture of single core and multicore jobs compete for the same resources a non-optimal setup may impact on the overall job throughput. For example, a high priority whole-node job could block access to a compute node until all cores are available for its exclusive use. During this wait time it is feasible that single-core jobs could have been executed and completed on the host marked for use (a process known as backfilling) if the lifetime of the whole-node job was known in advance to the scheduler.

The only efficient use of the resources with low submit times is the use of one job per machine. When the jobs are finished, the whole node is then immediately free to accept the next in the queue, and submit times remain proportional to queue length. Although this was a relatively simple solution to implement it may not be the most suitable solution for every site, especially where the rate of whole-node jobs submitted is small. In low-usage periods any resources marked for whole-node use will remain idle and could have been used to process single-core jobs instead.

At this time several queues have been set up for AthenaMP use. Since AthenaMP is not in full data production the number of resources dedicated to these queues is relatively low. AthenaMP jobs have been run on each of these test queues and shown to be using the correct number of processes for the target worker nodes. Once run the status of the jobs is correctly determined, log files and the associated data produced by each job are copied to central storage resources. Job success and consequent data transfer is monitored by the ATLAS Panda job management system.

One difficulty with the above job submission procedure is that many sites need to manage large sets of non-homogeneous resources, and usually do not have the same number of CPU cores per machine in use throughout a single queue. Defining and managing special queues for MP use becomes difficult for sites in these cases and the queues will be defined on a set of machines with variable numbers of CPU cores.

In order to improve CPU efficiency of AthenaMP jobs it is necessary to adjust their size, i.e. the number of events to be processed by a given job, in accordance with number of CPU cores available on a given node. By running longer multi-process jobs we minimize the time the job spends in its sequential part and by this way we achieve better overall event throughput.

4. Validation of AthenaMP

The validation process of AthenaMP took several months of coordinated effort and which will now continue in step with the future Athena release cycle. At the first stage we had to make sure that AthenaMP can be used for running different types of reconstruction jobs, that there were no critical bugs and crashes and also the output produced by AthenaMP is sensible. After reaching a good level of stability we started to experiment with sending AthenaMP jobs to whole-node queues on the Grid and also we ran a series of large scale tests at the CERN Tier-0 cluster.

At the same time we started to perform more detailed comparisons of output files produced by AthenaMP and by serial Athena reconstruction jobs. For such comparison tests we usually ran reconstruction over small number of events and compared outputs bit by bit. After fixing all sources of differences seen between outputs of serial and multi-process jobs, we now plan to run AthenaMP reconstruction over large event samples and pass the results to ATLAS physics validation groups for detailed studies.

4.1. Large-scale tests of AthenaMP at CERN Tier-0 cluster

As the part of AthenaMP validation process we ran two series of large scale tests at the CERN Tier-0 cluster. The main goal of these tests was to ensure no disruptive failures are observed and to compare parallel and sequential reconstruction performance characteristics. In total we ran over 1.1K AthenaMP reconstruction jobs, which processed in total around 2M events. A few minor problems which were not known before were promptly fixed in advance of the second round of testing. The resulting performance characteristics were collected from LSF logs.

Both test rounds demonstrated that parallel reconstruction is considerably slower than its MJ counterpart. Total wall clock times for both modes of running are collected in table 1.

Table 1. Total wall clock times of sequential (MJ) and parallel (MP) Athena reconstruction jobs measured during two rounds of tests at CERN Tier-0 cluster.

	Athena MJ	AthenaMP	Athena MJ/AthenaMP
Round 1	173 core-days	360 core-days	0.48
Round 2	72 core-days	106 core-days	0.68

The speedup of AthenaMP reconstruction observed in the second round of testing was achieved by switching to the fast merger utility for POOL output files. The remaining difference between parallel and sequential reconstruction performance was due to both series of tests ran MP and MJ jobs of the same size, thus while every single parallel job ran faster than its sequential counterpart the overall observed performance degradation of parallel jobs was a direct consequence of Amdahl's law [5]. The only way to make AthenaMP competitive to the sequential reconstruction in terms of event throughput is to minimize the contribution of the serial part – initialization, merging, finalization – to the total wall clock time of the parallel job. In order to achieve that we need to extend AthenaMP job size, which means running over multiple input event files instead of just one for a single AthenaMP reconstruction job.

5. Performance studies

In order to study AthenaMP performance characteristics we ran a series of reconstruction jobs on dedicated hardware: dual quad-core 2.27 GHz Intel Xeon L5520 processors with Hyper-Threading enabled and with a total memory of 24GB. We ran either single AthenaMP job with N workers or N independent sequential Athena jobs.

Figure 2 shows maximal memory usage on the machine during the event processing loop. It does not take into account memory access spikes observed at finalization stage of worker processes. This figure demonstrates the effect of late forking on memory sharing optimization in AthenaMP jobs. In typical Athena reconstruction jobs a large memory chunk gets allocated after the initialization phase once the first event is processed. At this point the job loads and caches a large amount of data read from ATLAS conditions database. If we fork sub-processes after the first event then parent process memory is shared between AthenaMP workers, which results in a significant reduction in total memory footprint compared to running multiple instances of Athena. For certain types of reconstruction jobs we can share up to 80% memory pages among event worker processes.

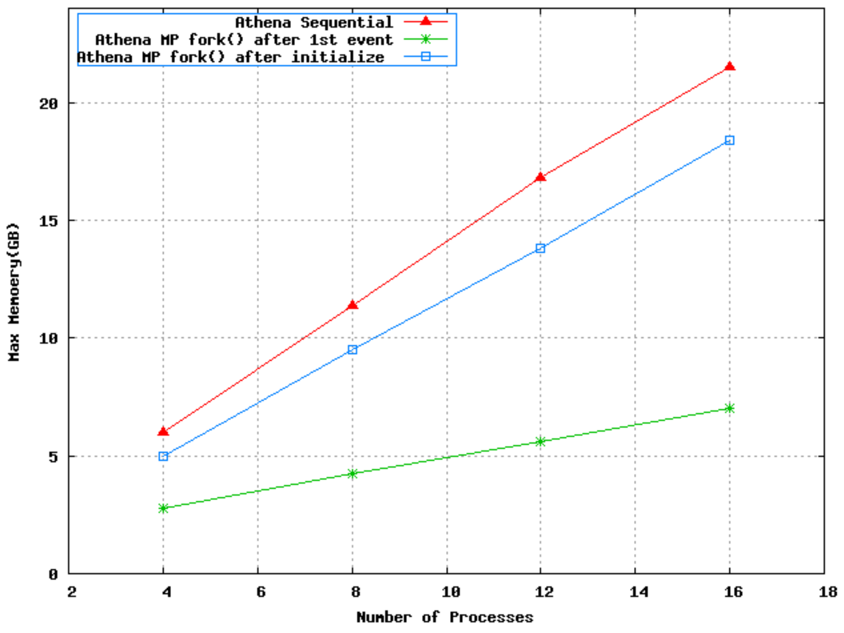

Figure 2. Maximal memory consumption seen on a dedicated machine during AthenaMP reconstruction event loop. Demonstrates the effect of late forking on memory sharing between worker processes.

6. Further improvements of AthenaMP performance

As already mentioned in section 4.1, in order to compete with event throughput of N individual sequential Athena reconstruction jobs executed in parallel, it is necessary to increase the size of AthenaMP jobs (i.e. increase the number of events to be processed by the jobs), because by doing this we minimize the fraction of total wall clock time spent in the sequential part of the job. Increasing the job size may seem to be fairly easy to achieve – one just needs to feed reconstruction with multiple input files instead of one, however in practice this will not be as simple when it comes to running such large jobs on the ATLAS distributed production system. In order to cope with this requirement the pilot process on a worker node will have to determine the number of processes to use depending on available system resources. Once determined the pilot will request a job of the appropriate size to be executed with that number of processes. Such mode of operation is currently under development.

An important area of AthenaMP, which requires long term development, is related to handling of I/O operations. The current implementation, when worker processes read and write files independently, does not scale with the number workers and in order to improve this situation we plan to develop two types of specialized I/O workers – Event Source (reader) and Event Sink (writer), which will communicate events with event workers via shared memory.

By introducing such processes we expect to reduce disk access and overall memory footprint, as, for example, all reading will be done in one single process and we will avoid duplicating ROOT [6] buffers. Also the combined event reader will reduce the number of open file handles. Specialized I/O processes will free event workers from dealing with I/O operations and by this way improve overall CPU performance of AthenaMP jobs. Finally, by introducing an Event Sink process we eliminate the need of output file merging. Events for writing will be provided to the Event Sink by event workers via dedicated shared memory segments and the actual writing to the file will be done only by a single process.

7. Summary

After initial development and validation phase AthenaMP is entering the stage when it can be used in large scale test campaigns, whose results will be handed over to ATLAS physics groups for further validation. By running reconstruction jobs in parallel using AthenaMP we can achieve much reduced overall memory footprint than by running the same number of independent serial jobs. The fast merger utility for output POOL file merging has greatly improved the event throughput of AthenaMP jobs, however in order to compete in CPU efficiency with serial jobs we need to increase AthenaMP job sizes by running them over multiple input files instead of just one. After passing the validation procedures we plan to start using AthenaMP widely on the ATLAS production system.

In order to be able to run AthenaMP jobs on the Grid special whole-node production queues have been configured at CERN and at several Tier-1 and Tier-2 sites. The necessary configuration changes have been applied to the batch system scheduler at each participating site to make computing resources available for whole-node jobs. AthenaMP use on the Grid poses a set of unique difficulties and there are several concerns related to job scheduling and efficiency, which are currently being addressed.

The optimal usage of available memory resources, good level of stability and good agreement between parallel and sequential reconstruction results makes AthenaMP a viable candidate to become a default mode for running large scale productions at CERN Tier-0 and on the Grid.

References

[1] Binet S, Calafiura P, Snyder S, Wiedenmann W and Winklmeier F 2010 Harnessing multicores: strategies and implementations in ATLAS *J. Phys.: Conf. Ser.* **219** 042002

[2] Mato P 1998 Gaudi - architecture design document *Tech. Rep.* LHCb-98-064 Geneva

[3] The multiprocessing module http://docs.python.org/library/multiprocessing.html

[4] POOL – persistency framework http://pool.cern.ch

[5] Amdahl G 1967 Validity of the single processor approach to achieving large-scale computing capabilities *AFIPS Conference Proceedings* (30) pp 483-85

[6] ROOT – data analysis framework http://root.cern.ch