
Panel: What are the necessary ingredients
for scalable OpenMP programming?

Sanjiv Shah
KAI Software Laboratory
Intel Corporation

Panel Questions

1) Large shared memory systems will have non uniform memory access. How do we cope with it?

Memory placement will be an issue, for ccNUMA systems as well as for software distributed shared memory systems. Do we need additional directives, or can this be handled efficiently enough by the underlying system?

(This topic has already been discussed often, this time I want to take the opportunity to hear and discuss the opinions from the application side.)

2) Do we need more asynchronicity in the OpenMP API in order to be able to build scalable applications?

Task queues, nested parallelism, more flexible and elegant synchronization primitives have been proposed, their validity has been demonstrated, yet except for nested parallelism - the compiler manufacturers are very hesitating to attack it - these elements did not yet find their way in the OpenMP API.

3) Can DSM-OpenMP be an alternative to MPI?

Several compilation systems for the usage of OpenMP on Clusters are under development. So far OpenMP even ignores the existence of ccNUMA systems. On the other hand HPF was not particularly successful in the past. Is Cluster-OpenMP a promising approach to implement parallel applications on a poor man's parallel computer?

Large shared memory systems will have non uniform memory access. How do we cope with it?

- There is always a tradeoff between simplicity and ultimate performance
- Lesson from Java
 - The commercial software industry is starting to lean very heavily toward increased simplicity and reliability and willing to give up performance
- OpenMP's strength is it's simplicity
- Another lesson from the past: rushing into bad programming models because of hardware architecture *du jour* is not sustainable
- Are the existing specifications are not sufficient for NUMA? Is it possible that NUMA tuning is all about “optimizing” implementations to co-locate data & threads, and “training” users to pay more attention to the data touched by threads?

Optimize implementations, train users and let the hardware & operating systems mature!

Can DSM OpenMP be an alternative to MPI?

- In general, comparing the performance of DSM systems to MPI is a futile exercise
 - Would you compare the performance of Java to assembly language routines, or C++ to Fortran kernels?
- Depending upon your needs, DSM OpenMP can be a very viable alternative.
- Pros
 - Simplicity over ultimate performance
 - Developer productivity gains
- Caution: many problems remain for DSM systems to be widely adopted and depend upon markets being targeted

For the right applications, DSM OpenMP should trounce MPI when you look at ROI over the life of software projects

Do we need more asynchronicity in the OpenMP API in order to be able to build scalable applications?

- True asynchronicity would allow splitting of all transactions and have handles for each
 - Results in loss of “sequential consistency” and simplicity
- What are the full software lifecycle costs and benefits of true asynchronicity?
- Some of the elegant synchronization and WorkQueuing proposals add the right amount of asynchronicity and manage to avoid the potholes

For OpenMP to expand its reach to more applications, standardizing the proposals seems crucial. But let's avoid the potholes of true asynchronicity!

Other Relevant Issues

- Nested OpenMP is far from mature
 - Many parts of the specifications are not well defined
- Affinity models become far more important

Summary

- Scalability can be considered in many dimensions
 - Single System: rushing into NUMA programming models has proved to be futile. Let the systems mature! Improve nesting and affinity controls, optimize implementations, train users
 - Cluster: DSM solutions can be scalable to the order of a 100 processors. Should this be standardized?
 - Application Domain Scalability: WorkQueuing helps tremendously, may help scalability. Why is this not standard?