Abstract

- the papers abstracts a number of considerations that must be observed in new UPC compiler developments, such as for the future IBM PERCS architecture
    - PERCS is supported in part by the DARPA under contract No. NBCH30390004

Introduction

A Quick UPC Overview

- under UPC, application memory consists of two separate spaces: a shared memory space and a private memory space.
- among the interesting and complementary UPC features is a wok-sharing iteration statement, known as upc_forall()
- the language offers a broad range of synchronization and memory consistency control constructs.
- among the most interesting synchronization concepts is the non-blocking barrier, which allows overlapping local computations with synchronization

Background on PERCS and UPC

- DARPA HPCS program is concerned with the development time of applications as well, and aims at significantly reducing the overall time-to-solution.
- PERCS is using innovative techniques to address lengthening memory and communication latencies

UPC Compilers on Distribute Memory Systems

Low-Level Library Implementations

- In some compiler implementations this can be costly to the point that makes an access into a thread ’s private space orders of magnitude faster than accesses into the thread ’s local shared space, even though both spaces can be mapped into the same physical memory.
- the fact that UPC compilers are still maturing and may not automatically realize that local accesses can avoid the overhead of shared address calculations
- An important emulation of effective access to local shared data is privatization, which makes local shared objects appear as if they were private. This emulation is implemented by casting a local shared pointer to a private pointer and using the private pointer for accessing the local shared objects as if they were private.

Aggregation and Overlapping of Remote Shared Memory Accesses

- Affinity analysis at compile time may reveal what remote accesses will be needed and when.
- Therefore, one other optimization is aggregating and prefetching remote shared accesses, which as shown in Table 2 can provide substantial performance benefits.
- split-phase barriers, to hide synchronization cost
- as source to source automatic compiler optimizations.
- The N-Queens problem does not get any benefit from such optimizations
- which does not require significant shared data.
- that N-Queens and embarrassingly parallel applications that do not require extensive use of shared data will not be adversely affected by such optimizations.
- The Sobel Edge Detection performance on the NUMA machine: This case illustrates the importance

of these optimizations.

Low level communication optimizations

- Low level communication optimizations like Communication and Computation Overlap and Message Coalescing and Aggregation have been shown in the GASNET interface [CHEN03].

UPC OPTIMIZATIONS AND MEASUREMENTS ON DISTRIBUTED SHARED MEMORY SYSTEMS

Shared Address Translation

- the memory model translation to virtual space can be quite costly
- it is clear that the address translation overhead is quite significant as it added more than 70% of overhead work in this case.

UPC Work-sharing Construct Optimizations

- UPC features a work-sharing construct, upc_forall, that can be easily used to distribute independent loop body across threads, figure 7
- High quality compiler implementations should be able to avoid these extra evaluations.
- Evaluating the affinity field will introduce additional overhead.
- “ overhead-free ” for loops

Performance Limitations Imposed by Sequential C Compilers

- Recent studies [ElGH05] over the distributed shared memory NUMA machines, clusters, and Scalar/vector architectures have shown that the sequential performance between C and Fortran can greatly differ.
- The measurements show that sequential C is generally performing roughly two times slower than Fortran.
- Especially for the scalar/vector architecture in which the vectorized codes usually perform much faster than those executed only in the scalar units.
- with the use of proper pragma(s), similar levels of performance were obtained.

LESSONS LEARNED AND PERCS UPC

- UPC implementation on future architectures such as PERCS must take advantage of the performance studies on other architectures.
- four types of optimizations
  - Optimizations to Exploit the Locality Consciousness and other Unique Features of UPC,
  - Optimizations to Keep the Overhead of UPC low
  - Optimizations to Exploit Architectural Features
  - Standard Optimizations that are Applicable to all Systems Compilers
- creating a strong run-time system that can work effectively with the K42 Operating System
- K42 is able to provide the language run-time system with the control it needs
- The run-time system will be extremely important in future architectures

SUMMARY AND CONCLUSIONS

- UPC is a locality-aware parallel programming language.
- UPC can outperform MPI in random short accesses and can otherwise perform as good as MPI.
- UPC is very productive and UPC applications result in much smaller and more readable code than

MPI.

- For future architectures such as PERCS, UPC has the unique opportunity of have very efficient UPC implementations as most of the pitfalls and obstacles are now revealed along with adequate solutions.