

Portable Support and Exploitation of Nested Parallelism in OpenMP

Panagiotis E. Hadjidoukas Laurent Amsaleg

IRISA/INRIA, Rennes, France

TEXMEX group

{phadjido,lamsaleg}@irisa.fr

EWOMP 2004

Stockholm, Sweden, 18-21 Oct. 2004



Overview

- OpenMP runtime support (compiler/library developers)
 - Portable implementation of NthLib
 - Support for nested parallelism
 - Integration of NthLib into open source OpenMP compilers
 - Experimental implementation of the workqueuing model
- Parallel implementation of a data clustering algorithm
 - Parallelization using OpenMP
 - Exploitation of nested parallelism
 - Benchmark / Real application

Goal

- General implementation of the NANOS execution environment
 - On shared memory machines
 - NANOS project (1996-1999)
 - <http://www.ac.upc.es/nanos>
 - On clusters
 - POP project (2001-2004)
 - www.cepba.upc.es/pop
- Main components
 - Compiler (NANOS)
 - Runtime Library (NthLib)
 - Multiprogramming module (CPU Manager)
- General
 - Portability
 - Any open-source OpenMP compiler
 - Several OpenMP implementations (Intone, Omni, OMPi, ORC)

Portability

- Two level thread model
 - User / Kernel level threads
 - Additional kernel threads just introduce overheads and cannot be controlled easily
- Portability Issues
 - Kernel Threads: POSIX Threads API
 - Synchronization: POSIX Threads API (or custom)
 - User-level threads: setjmp/longjmp or ucontext
- UthLib: Portable non-preemptive user-level thread package
 - Based on State Threads, derived from NPRL (Mozilla)
 - GPL/MPL license
 - Available at: <http://www.hpclab.ceid.upatras.gr/~peh>

Features of NthLib

- A stack should be always created
 - How do you know that you are at the inner level?
- Lazy stack allocation policy
 - Minimal user-level thread migrations
 - Minimal memory consumption
 - Stack reuse (per virtual processor)
 - Stack handoff (optimization)
- Efficient management of nested parallelism
 - All-to-all to scheme that favours locality
 - Portable Stealing (compared to Omni/ST)
- Inherent runtime support for workqueuing
 - Recursive parallelism
- Preemption (computational codes)

Spawning parallelism (Omni/NthLib)

```
void _ompc_do_parallel_main (int nthds, cfunc f, void *args)
{
    struct nth_desc *nth, *master;

    ...
    master = nth_self();
    nth_depadd(master, n_thds+1);
    for( i = 0; i < n_thds; i++ ){
        nth = nth_create_ls(f, 0, master, 1, args);
        ...
        nth->parent = master;
        nth->num = i;
        nth->num_thds = n_thds;
        if (self->parent == NULL)
            nth_to_lrq_end(i % lwps, tp);
        else
            nth_to_lrq(self->vp_id, tp);
    }
    nth_block();
    ...
}
```

Spawning parallelism (Omni/NthLib)

```
void _ompc_do_parallel_main (int nthds, cfunc f, void *args)
{
    struct nth_desc *nth, *master;
    struct nth_desc snth[MAX_THREADS];
    ...
    master = nth_self();
    nth_depadd(master, n_thds+1);
    for( i = 0; i < n_thds; i++ ){
        nth = nth_create_ls_opt(&snth[i],f,0,master,1,args);
        ...
        nth->parent = master;
        nth->num = i;
        nth->num_thds = n_thds;
        if (self->parent == NULL)
            nth_to_lrq_end(i % lwps, tp);
        else
            nth_to_lrq(self->vp_id, tp);
    }
    nth_block();
    ...
}
```

Parallel Data Clustering

- CURE – Hierarchical (agglomerative) algorithm
- Data Structures
 - Array of records (vectors of dimension 2,...)
 - Entry for each cluster
 - Centroid
 - Representative points
 - Size
 - Index of closest cluster and distance to it
- Distance
 - Between vectors → Euclidean ($n=2$) or higher norm
 - Between clusters
 - Distance of their centroids
 - Minimum distance of their representatives

Clustering Algorithm

1. Load records (read from file)
2. Initialization (every record is a cluster of size 1)
 - Compute distances and find nearest neighbors for all clusters
3. Clustering: Perform hierarchical clustering until the predefined number of clusters (k) has been computed
 - Find nearest clusters and merge them
 - Update distances and nearest neighbors
4. Output the centroid and the representative points for each cluster
5. Assign records to clusters

Parallel CURE – Initialization phase

```
find_nearest_neighbor(int i, int *idx, double *dist
{
    min_dist = +∞, min_idx = -1;
    for (j = 0; j < i; j++) {
        - if (entry j has been invalidated) continue;
        - if ((d = compute_dist (i, j)) < min_dist)
            { min_dist = d; min_idx = j };
    }
    *idx = min_idx; *dist = min_dist;
}

init_nearest_neighbors ()
{
    for (i = 0; i < npat; i++)
        find_nearest_neighbor(i, &index[i], &dist[i]);
}
```

Parallel CURE – Update phase

```
find_nearest_neighbor(int i, int *idx, double *dist
{
    min_dist = +∞, min_idx = -1;
    for (j = 0; j < i; j++) {
        - if (entry j has been invalidated) continue;
        - if ((d = compute_dist (i, j)) < min_dist)
            { min_dist = d; min_idx = j };
    }
    *idx = min_idx; *dist = min_dist;
}

update_distances(int pair_low, int pair_high)
{
    for (i = 0; i < N; i++) {
        - if (entry i has been invalidated) continue;
        - if (entry i had neighbor pair_low or pair_high)
            find_nearest_neighbor(i, &index, &dist);
        - else if (pair_high < i)
            if ((d = compute_dist(pair_low, i)) < dist[i])
                { index[i] = pair_high; dist[i] = d; }
    }
}
```

Issues

- Several levels of parallelism
 - `update_distances`
 - `find_nearest_neighbor`
 - distance between clusters (number of representatives)
 - distance between vectors
- We search for the nearest neighbour
 - Some kind of reduction (minimum distance **AND** index)
 - Spawn OpenMP threads (`omp parallel`)
 - Execute the loop (`omp for nowait`)
 - Compute nearest neighbour locally in each thread
 - The threads update the global result (variable) that resides in their parent's stack (`omp critical`)

Scalability

- No data dependencies and good locality
- Distances and Indexes can be computed locally (static)
- Only two entries are written, the rest are only read
- No linear speedup due to memory bandwidth
- How can we handle this on shared memory?
- OpenMP on SDSM

Demonstration

- Library/OpenMP runtime support
 - Study the implementation issues
- Applications
 - Study the code of PCURE / Compile and run it
 - Demonstrate execution of recursive parallelism
 - Try to compile your application

OpenMP environment

- Primary development platform: Linux (IA32)
 - Pentium Xeon, Linux (IRISA)
 - Opteron, Linux (KTH)
 - SUN, Solaris (KTH)
 - IBM, AIX (CEPBA)
 - SGI, IRIX (CEPBA)
- Compiler: OMPi (www.cs.uoi.gr/~ompi)
 - OpenMP v. 2.0
 - Nested parallelism by NthLib
 - Portable (written in C)
 - Omni requires Java