Maximizing performance of irregular applications on multithreaded, NUMA systems

Guojing Cong and Sophia Wen
IBM TJ Watson Research Center

IA ³, Nov 17, 2013, Denver
Introduction

- Available resources and communication latency for a group of logical processors are determined by their relative position in the hierarchy of chips, cores, and hardware threads.

- For applications designed under a flat programming model (UPC, MPI, OpenMP, pthreads), we want to deploy (map/optimize) them to the hierarchy of target architecture for best performance.
Choose a mapping that balances communication latency and available resources for an application
  - As much as 5.4 times difference in performance is observed
  - Default mapping is the worst mapping for irregular applications
  - A metric to detect the mapping problem

Optimizations
  - Techniques for optimizing geographical locality
  - Unified approach
IBM P755, 4 POWER7 chips, and each chip has 8 cores; Each core capable of four-way simultaneous multithreading (SMT) 12 execution units including 2 fixed-point units and 2 load/store units per core
Each core has a 32KB private L1 and a 256KB private L2. The on-chip shared 32MB L3 is comprised of 4MB local L3 regions from the eight cores
The POWER7 core switches among ST (single thread), SMT2, and SMT4 dynamically
Impact on resource contention

One thread pinned to logical cpu 0, the other changes from 1 to 127. They run the same loop of multiply-add instructions.
Impact on communication

The two threads act as producer/consumer using flags.
Mapping strategies

- $\frac{c!}{(c-t)!}$ possible mappings for $t$ software threads with $c$ (\(t \leq c\)) logical CPUs
- on P755 128 threads, $4 \times 8 \times 4$ grid
- we consider three modes
  - chipfirst: saturate first the chip dimension, then the core dimension, and finally the thread dimension
  - threadfirst: saturate first the thread dimension, then the core dimension, and finally the chip dimension
  - corefirst: saturate first the core dimension, then the chip dimension, and finally the thread dimension
Performance under three mappings
Performance under three mappings

### SP (CLASS B)

![Graph showing performance under SP mapping for CLASS B](image)

- **Time (seconds) vs. #threads**
- **Legend:**
  - **thread**
  - **chip**
  - **core**
  - **default**

### MG (CLASS B)

![Graph showing performance under MG mapping for CLASS B](image)

- **Time (seconds) vs. #threads**
- **Legend:**
  - **thread**
  - **chip**
  - **core**
  - **default**

### CG (CLASS B)

![Graph showing performance under CG mapping for CLASS B](image)

- **Time (seconds) vs. #threads**
- **Legend:**
  - **thread**
  - **chip**
  - **core**
  - **default**

### IS (CLASS B)

![Graph showing performance under IS mapping for CLASS B](image)

- **Time (seconds) vs. #threads**
- **Legend:**
  - **thread**
  - **chip**
  - **core**
  - **default**
Remoteness for NAS and SPEC benchmarks

Solution: use thread first binding for applications with large $r$
Optimizing geographical locality

- bring data close to processing by matching data distribution with data accesses
  - Replication: replicate shared data that are not frequently updated
  - Simple) data (re)-distribution – either statically or dynamically
  - Mostly work for regular codes, serious limitations for irregular codes on large data
    - replication does not scale
    - static distribution oftentimes does not work for complex access pattern
    - dynamic redistribution involves a profiling run
Optimizing geographical locality

- bring data close to processing
- match data distribution with data access pattern
  - bipartite graph partitioning to find the best distribution
  - access permutation to find the best access scheduling
- Locality optimization
Bipartite graph partitioning

\[ Y = \{ y_1, y_2, \cdots, y_k \} \text{: set of shared data} \]
\[ X = \{ x_1, x_2, \cdots, x_l \} \text{: set of program structures that access } Y \]

Let \( V = X \cup Y \)

\((x_i, y_j) \in E, 1 \leq i \leq k, 1 \leq j \leq l\), if \( x_i \) accesses \( y_j \)

\( w(x_i, y_j) \) is the number of times \( x_i \) accesses \( y_j \)

Partition both \( X \) and \( Y \) into \( p \) equal-sized blocks \( X_1, X_2, \cdots, X_p \)

and \( Y_1, Y_2, \cdots, Y_p \), respectively, so that the cut among \( V_1 = X_1 \cup Y_1, V_2 = X_2 \cup Y_2, \cdots, V_p = X_p \cup Y_p \)

\[ \sum_{1 \leq s < t \leq p} \sum_{u \in V_s, v \in V_t} w(u, v) \]

is minimized.
Example: match accesses to distribution

CC(El, D): El input edge list, D[i] is the current component vertex i belongs to

for 1 \leq i \leq m in parallel do
  if D[El[i].u] < D[El[i].v] then
    D[D[El[i].v]] ← D[El[i].u]
  end if
end for

for 1 \leq i \leq n in parallel do
  while D[i] \neq D[D[i]] do
    D[i] ← D[D[i]]
  end while
end for
Permuting for CC

$u$ and $v$ determine whether $D[u]$ and $D[v]$ are remote.
Instead of redistributing $D$, permute accesses to $D$; that is, permute the edges in $E_l$ to reduce remote accesses to $D$. It is impossible to make both $D[u]$ and $D[v]$ local for all $(u, v) \in E$.

Sort $E_l$ with $E_l[i].u/(n/q)$ as key, $1 \leq i \leq m$, and $q$ is the number of chips.

After sorting, the $D[u]$s are mostly local.
Results: geographical optimizations

Data duplication – CG

Data permutation – CC
Unified approach

- From the perspective of a single thread, data from remote caches or memory are not much different from those in the local memory hierarchy except for longer access latency.
- Even with perfect geographical locality, irregular applications still have poor cache performance.

\[
r = \frac{\text{#remote accesses to shared memory}}{\text{#shared-memory accesses}} > \text{threshold}
\]

\[
r = \frac{\text{#cache misses}}{\text{#memory accesses}} > \text{threshold}
\]
Shared memory: \( \text{par\_access}(C, D, R, p, n, m) \)

if \( n \leq z \) then
    \( \text{return } C[i] \leftarrow D[R[i]], 1 \leq i \leq m \)
end if

{partition}
divide \( R \) and \( D \) into \( p \) blocks, \( R_1, \ldots, R_p \), and
\( D_1, \ldots, D_p \), of size \( s = m/p \) and \( w = n/p \),
respectively
for \( 1 \leq k \leq p \) in parallel do
    count sort \( R_k \) with key \( \frac{R_k[j]}{s} \) and keep original location
    of the \( j^{th} \) element in \( P_k[j], 1 \leq j \leq s \)
    partition \( R_k \) into \( p \) blocks \( R_{k,j}^l, 1 \leq j \leq p \), such that
    \( \forall r \in R_{k,j}^l, \frac{r}{s} = j \)
end for
for \( 1 \leq j \leq p \) in parallel do
    \( R_j' \leftarrow \bigoplus_{k=1}^p R_k^l \)
end for
{access}
for \( 1 \leq k \leq p \) in parallel do
    \( \text{par\_access}(S_k, D_k, R_k', p, n/p, |R_k'|) \)
end for

\{permute\}
for \( 1 \leq k \leq p \) in parallel do
    partition \( S_k \) into \( p \) consecutive blocks \( S_k^j, 1 \leq j \leq p \),
    such that \( |S_k^j| = |R_k^j| \)
end for
for \( 1 \leq k \leq p \) in parallel do
    \( S_k' \leftarrow \bigoplus_{j=1}^p S_k^j, 1 \leq k \leq p \)
end for
for \( 1 \leq k \leq p \) in parallel do
    if \( s \leq z \) then
        for \( 1 \leq j \leq s \) do
            \( C_k[P_k[j]] \leftarrow S_k'[j] \)
        end for
    else
        divide \( S_k' \) and \( P_k \) into \( s/z \) blocks, each of size \( z \);
        for \( 1 \leq j \leq s/z \) in parallel do
            sort \( S_k' \) by corresponding \( P_k \) value
        end for
        \( z\)-way merge sort blocks of \( S_k' \) into \( S_k'' \)
        for \( 1 \leq j \leq s \) do
            \( C_k[j] \leftarrow S_k''[j] \)
        end for
    end if
end if
return \( C \leftarrow \bigoplus_{k=1}^p C_k \)
$b$: memory bandwidth; $l$: memory latency;

before:

\[ T_M = m \left( l + \frac{1}{b} \right) \]

after:

\[ T_M = f(n, m) = \begin{cases} 
  z(l + 1/b) & n \leq z \\
  3pl + 3p^2l + \frac{p^2}{b} \\
  + m\left( \frac{5}{b} + \frac{\log_z m}{b} \right) + pl\log_z \frac{m}{p} \\
  + \sum_{i=1}^{p} f(n/p, m_i) & n > z
\end{cases} \]

Solving the recurrence yields

\[ f(n, m) \leq g(n) = (p \alpha + \beta) \log_p n + z(l + 1/b) \]

with $z \geq 2\sqrt{n}$, $\alpha = 3pl + 3p^2l + \frac{p^2}{b}$, $\beta = m\left( \frac{5}{b} + \frac{\log_z m}{b} \right) + pl\log_z \frac{m}{p}$, In order that $f(n, m) < m(l + 1/b)$ we require at least

\[(3p + 6p^2l + 3p^3 + p\log_z \frac{m}{p})l\log_p n + zl < ml \]

Roughly the network has to satisfy

\[ lb > \frac{4}{3} \left( 5 + \log_z \frac{m}{p} \right) \log_p n \]
Results

Power 6 – impact of virtual threads

Power 7, in log-log scale

More than 2× speedup
Results

Power 6 – impact of virtual threads

Power 7, in log-log scale

More than 2× speedup
Conclusions

- Deploying a multithreaded program onto current NUMA systems has significant performance impact
- NUMA specific optimizations oftentimes have serious limitations for irregular applications
- Our unified approach improve both cache and communication performance simultaneously