Towards Exploiting Data Locality for Irregular Applications on Shared-Memory Multicore Architectures

Cheng Wang

Advisor: Barbara Chapman

HPCTools Group,
Department of Computer Science,
University of Houston, Houston, TX, 77004, USA

May 17, 2016
Outline

1. What are irregular applications?
2. Sparse FFT - A case study of irregular applications
3. A padding algorithm to improve the data locality
4. Conclusion & Future work
The Reality of Parallel Computing ...
Why CPU Caching Matters?

- Memory has become the principle performance bottleneck
- Improve the cache utilization is the key to performance optimization

---

1Source: http://cs.uwec.edu/~buipj/teaching/cs.352.f12/lectures/lecture_08.html
Shared memory

- **On-chip**: (Last-level) cache shared by homo/hetero processors
- **Off-chip**: Main memory shared by homo/hetero processors
What are Irregular Applications?

Indirect array reference pattern

Commonly found in linked list, tree and graph-based applications

Poor data locality

Challenge esp. for shared-memory multicore architecture as cores compete for memory bandwidth

```plaintext
do i = 1, N
    ... = x[id * [i]]
end do
```
**Approach: Computation/Data Reordering**

- **Computation**
  - 1
  - 2
  - 3
  - 4

- **Data**
  - 1
  - 2
  - 3
  - 4

---

**Computation reordering**

1. 3
2. 1
3. 2
4. 4

**Data reordering**

1. 2
2. 3
3. 1
4. 4

---

*May 17, 2016*  
*Cheng Wang (cwang35@uh.edu)*
Challenges in Dynamic Irregularity Removal

1 Dynamic irregularity
   - Memory access pattern remains unknown until runtime and may change during computations
   - Previous work on compile-time transformations can hardly apply
   - Need for transformation at runtime

2 Runtime transformation overhead
   - Transformation overhead is placed on the critical path of the application’s execution
   - The benefits of improved data locality must outweigh the cost of the data layout transformation at runtime
1. What are irregular applications?
2. Sparse FFT - A case study of irregular applications
3. A padding algorithm to improve the data locality
4. Conclusion & Future work
**Sparse FFT**

1. A novel compressive sensing algorithm with massive application domains

2. Fourier transform is dominated by a small number of “peaks”
   - $\text{FFT}(O(n\log n))$ is inefficient

3. Compute the $k$-sparse Fourier transform in lower time complexity
   - $k$-sparse: no. of “large” coordinates at freq. domain
Sparse Data is Ubiquitous ...

Paper: [ISMRM’13]
MRS Sparse-FFT: Reducing Acquisition Time and Artifacts for In Vivo 2D Correlation Spectroscopy

Paper: [MOBICOM’12]
Faster GPS Via the Sparse Fourier Transform

1Slide based on http://groups.csail.mit.edu/netmit/sFFT/
Irregular Memory Access Pattern in Sparse FFT

- Randomly permutes the signal spectrum and bins into a small number of buckets
- Irregular memory access pattern

\[
\text{buckets}[i \% B] += \text{signal}[idx] \times \text{filter}[i]
\]
Parallel Sparse FFT

1. Modern architectures are exclusively based on multicore and manycore processors
   - e.g., Multicore CPUs, GPUs, Intel Xeon Phi, etc.
   - Nature path to improve the performance of sFFT through efficient parallel algorithm design and impl.

2. Standard full-size FFT has been well studied and implemented
   - FFTW, cuFFT, Intel MLK, etc..
   - Highly optimized for specific architectures

3. We are the first such effort of high-performance parallel sFFT implementation

---

• **cuFFT**: full-size FFT library on Nvidia GPUs
• The MIT seq. sFFT is slower than cuFFT
• cusFFT is **5x** faster than PsFFT, **25x** vs. the seq. sFFT
• cusFFT is up to **12x** faster than cuFFT
The seq. sFFT is slower than cuFFT
PsFFT is faster than cuFFT until $k = 3000$
cusFFT is faster than cuFFT until $k = 41,000$
1 What are irregular applications?
2 Sparse FFT - A case study of irregular applications
3 A padding algorithm to improve the data locality
4 Conclusion & Future work
Rethink the Consecutive Packing (CPACK) Algorithm

CPACK: A greedy algorithm which packs data into consecutive locations in the order they are first accessed by the computation.

Data access order: 9, 23, 103, 23, 67, 23, 67
7 cache misses

First-touch policy packs (9,23) together
Not optimal

Original
Data reordered by CPACK

CPACK
Computation
Data
1 2 3 4 5 6 7
9 ... 23 ... 67 ... 103

Computation
Data
1 2 3 4 5 6 7
9 23 67 103

6 cache miss
Rethink the Consecutive Packing (CPACK) Algorithm

Affinity-conscious data reordering ...

- CPACK does not consider data affinity (i.e., how close the nearby data elements are accessed together)
- Packs (23,67) rather than (9,23) should yield better locality
1 Finding an optimal data layout is a NP-complete problem\textsuperscript{1}

2 No “best” data reordering algorithm that works in general

3 \textbf{Implicit constraint:} Each data entry has only one copy in the transformed format

4 The complexity can be significantly reduced if more space is allowed to use

\textsuperscript{1}E. Petrank and D. Rawitz. 2002. The hardness of cache conscious data placement, POPL '02)
**CPACKE Algorithm**: Extends the CPACK by creating duplicated copies of each repeatedly accessed data entry.

**Advantage**: Better locality than CPACK

**Disadvantage**: Slight space overhead
• Applies the CPACKE to the perm+filter stage in sFFT
• Improves the performance by 30% for the irregular kernel
• Improves the overall performance of PsFFT by 20%
1 A padding-based algorithm improving the data locality of irregular applications
2 Improves the performance of sFFT by 30%
3 Future work
   - Evaluate with more irregular applications
   - Evaluate with other data/computation reordering algorithms
   - Let compiler generate the transformed code automatically