Asynchronous Parallel Disk Sorting

Roman Dementiev and Peter Sanders

Max-Planck-Institut für Informatik,
Saarbrücken, Germany

Thanks to: A. Crauser, D. Hutchinson, L. Kettner,
S. Mitra, A. Morton, N. Rajput, and J. Vitter
Motivation & Related Work

- Sorting is important for EM problems
- Disks are cheaper than computers
- Prefetch buffers for load disk balancing and overlapping have been studied but no results which guarantee
  - overlapping during merging of runs
  - asymptotically optimal I/O volume
- **Here**: algorithm & implementation
  - I/O cost $\rightarrow$ lower bound
  - guarantees almost perfect overlapping between I/O and computation
Motivation & Related Work

- [ChaudhryCormen’01’02] impl. of distributed memory, parallel disk
- Theory: [BarvGroVit’97] → [SandEgnKorst’00] → [HutchVitt’01] → [HutchSandVitt’02]
- Implementations: [BarveVitter’99] Differences:
  - Optimal prefetching algorithm
  - Overlapping of I/O and computation
  - Completely asynchronous implementation
  - Part of <stxxl> library, compatible with STL. Use as simple as:

```cpp
stxxl::vector<my_type> v;
long long int i=vec_size; // 'long long int' is a 64-bit data type
while(i--)
    v.push_back(f(i)); // fill vector with some values
stxxl::sort(v.begin(), v.end()); // sort
// check order using STL predicate
assert(std::is_sorted(v.begin(), v.end()));
```

[DementievSanders’03]
These algorithms do not guarantee overlapping
External Multi-way Merge Sort

These algorithms do not guarantee overlapping

\[ \text{N} \rightarrow \text{input size} \]
\[ \text{M} \rightarrow \text{main memory size} \]
\[ \text{B} \rightarrow \text{block size} \]
\[ \text{D} \rightarrow \text{number of disks} \]
Multi-way Merge Sort with Overlapped I/Os

Theorem 1: If I/O and computation can be overlapped, N elements can be sorted in time

\[ T_{\text{formruns}} + \left| \log_{\Theta(M/B)} k' \right| \cdot T_m \]

where \( T_{\text{formruns}} = \max \left( k' T_{\text{sort}} \left( \frac{N}{k'} \right), T_{\text{I/O}}(N) \right) + T_{\text{start up}} \) is the time needed for run formation,

\[ T_m = \max \left( T_{\text{I/O}}(N), T_{\text{merge int}}(N) \right) + T_{\text{start up}} \] is the time needed for merging

\( k = \Theta(M/B) \) : # sequences,
\( k' = O(N/M) \) : total # runs.
Overlapping Run Formation

- **Easy:**

  ![Diagram of run formation]

  **Corollary 2:** Input of size $N$ \(\Rightarrow\) sorted runs of size $\sim M/2$ in time

  \[
  \max \left( 2T_{\text{sort}} \left( \frac{M}{2} \right) \frac{N}{M} , T_{\text{I/O}}(N) \right) + T_{\text{startup}}
  \]

  \(2M\) in the paper
Simple External Merging

- Smallest element of a block is a trigger

![Diagram of merging process with k sorted runs]
Simple External Merging

- Smallest element of a block is a **trigger**

![Diagram showing Simple External Merging process](image)

**merger**

**k sorted runs**

**sort pairs** `<trigger,block_id>`

**prediction sequence** $\sigma$

**k-merger**
Overlapping I/O and Merging

- Prediction of delay between issuing two reads is not easy:

```
merger

1^{B-12} 3^{B-14} 5^{B-16} ...

1^{B-12} 3^{B-14} 5^{B-16} ...

...

1^{B-12} 3^{B-14} 5^{B-16} ...

\text{\{k\}}
```

R. Dementiev and P. Sanders: Asynchronous Parallel Disk Sorting
Overlapping I/O and Merging

- Prediction of delay between issuing two reads is not easy:

- Solution:
Overlapping I/O and Merging

- Prediction of delay between issuing two reads is not easy:

- Solution:
Overlapping I/O and Merging

- Prediction of delay between issuing two reads is not easy:

- Solution:
Overlapping I/O and Merging [contd.]

- **Theorem 4**: Merging $k$ sequences with a total of $N'$ elements takes time
  \[
  \max\left(\frac{2LN'}{DB}, \ell N'\right) + T_{\text{startup}}
  \]
  - $\ell$: time to produce one element of output
  - $L$: time to input or output $D$ arbitrary blocks

- Our I/O thread strategy:
  - $\geq DB$ elements in write buffer $\implies$ output step
  - $< DB$ elements in write buffer AND $D$ overlap buffers avail. $\implies$ input step

- To prove Theorem 4 we distinguish two cases
  - I/O bound case: $2L \geq DB$?
  - Compute bound case: $2L < DB$?
Compute Bound Case

- Lemma 6: In compute bound case after \( k/D + 1 \) steps, the merging thread never blocks until all elements are merged.

- Notation:
  - \( \omega \): # elem. in write buffer
  - \( r \): # of elem. in the overlap and merge buffers

- Our I/O thread strategy:
  - \( \geq \) DB elements in write buffer => output step
  - \( < \) DB elements in write buffer AND D overlap buffers avail. => input step
I/O Bound Case

- **Lemma 7:** In I/O bound case the I/O thread never blocks until all input blocks are fetched.

- Our I/O thread strategy (the same):
  - \( \geq \) DB elements in write buffer \( \Rightarrow \) output step
  - \(< \) DB elements in write buffer AND D overlap buffers avail. \( \Rightarrow \) input step

- **Proof:** similar to compute bound case
Multi-head Model $\Rightarrow$ Multi-disk Model

- Randomized Cyclic Allocation [VitHut’01] makes accesses in $\sigma$ balanced

- Optimal prefetching from [HutchSanVitt’01, SanEgnKor’00]

- **Corollary 8**: For any $\varepsilon > 0$, prefetch buffer of size $m = \Theta(D/\varepsilon)$ merging $k$ sequences with a total $N'$ elements can be implemented to run in time

$$\max\left(\frac{2LN'}{(1-\varepsilon)DB}, \ell N'\right) + T_{\text{startup}}$$
Multi-head Model ➔ Multi-disk Model

- Randomized Cyclic Allocation [VitHut’01] makes accesses in $\sigma$ balanced

- Optimal prefetching from [HutchSanVitt’01, SanEgnKor’00]

- **Corollary 8**: For any $\varepsilon > 0$, prefetch buffer of size $m = \Theta(D/\varepsilon)$ merging $k$ sequences with a total $N'$ elements can be implemented to run in time

$$\max\left(\frac{2LN'}{(1-\varepsilon)DB}, \ell N'\right) + T_{\text{startup}}$$
Implementation Issues

- Implementation (part of `<stxxl>` library)
  - Forming runs: Key sorting, efficient two passes MSD radix sort
  - Multi-way merging: [Sanders’00]
  - prefetch buffer + overlap buffer = read buffer
  - Asynchronous I/O (POSIX threads)
  - No superfluous copying
    - User blocks are transferred by DMA (O_DIRECT flag)
    - Buffers are passed by pointer between pools
Hardware

3000 Euro, 375 MByte/s, July 2002
Single Disk Performance

- LEDA-SM, TPIE
- 2 GB volume, 32-bit keys, runs of size 256 MB, g++ 3.2 -O6
- I/O bandwidth 45.4 Mbyte/s = 95% of peak bandwidth of the disk
Multiple Disk Performance

- 2 GB volume, 32-bit keys, runs of size 256 MB, g++ 3.2 –O6
- TPIE, LEDA-SM
  - Linux Soft-RAID 8X128KByte stripes
- Bandwidth of 315 MByte/s
Element Size

- 16GB volume, 32-bit keys, runs of size 256 MB, g++ 3.2 –O6
- $\geq 64 \Rightarrow$ merging is I/O bound
- $\geq 128 \Rightarrow$ run formation is I/O bound
- Small elements require special treatment
Optimal Prefetching

- 2D write buffers
- k+O(D) overlap buffers
- read buffers
- D blocks
- merging

Graph shows merging time in seconds against the number of read buffers.
Block Size

- Block sizes of several MB => good performance
- Leave room for read and write buffers
- Naive striping implies factor 8 block size reduction
  ⇒ Dramatic run time increase
Large Inputs

- One-pass, curves go up because of
  - Slower zones
  - Seek times
Summary

- Parallel disk sorting with high performance on state of art hardware with theoretical performance guarantees
- Bandwidth is no longer a limiting factor for external memory algorithms
- Price performance ratio can improve by adding disks