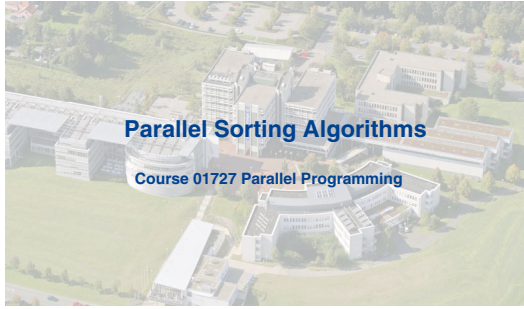


Ferdinand-Universität in Hagen Department of Mathematics and Computer Science



Parallel Sorting Algorithms
Course 01727 Parallel Programming

Parallelism and VLSI Group
Prof. Dr. Jörg Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Overview

- Why Parallel Sorting?
- Parallel Quicksort
- Bitonic Sort
- Parallel Merge Sort
- Summary

Slide 2 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Why Parallel Sorting?

- One of the most important subroutines
- Heavily investigated since >40 years
- Large data sets
- Looks quite sequential
- More difficult than numerics:
little computation, mainly control and data movement

Slide 3 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Why Parallel Sorting? – cont'd

- Lots of parallel algorithms
- Three representatives:
 - top-down/divide-conquer : quicksort
 - sorting network : bitonic sort
 - bottom-up : merge sort
- Concentrate on shared memory
- Hints for message passing
- Last two have been used on Cell BE processor

Slide 4 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort I

- Reminder: sequential quicksort

```

qsort(int a[n]){
  choose pivot a[i];
  alow = {all a[j] with a[j]<a[i]}; // partition array
  ahigh = {all a[j] with a[j]>a[i]};
  qsort(alow); qsort(ahigh); // divide
  a = concat(alow, a[i], ahigh); // conquer
}

```

- Complexity: $O(n \log n)$ on average

Slide 5 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort II

- Pivot can be chosen randomly
- Better: draw random sample of size $O(\sqrt{n})$
choose pivot as median
improves balance of alow to ahigh
- Pivot randomly attached to one of the partitions
Randomly to avoid continued disbalance
Attachment avoids separate treatment, e.g. in concat

Slide 6 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort III

- Partition implemented as reordering


```

left = 0; right = n-1;
do{
  while(a[left]<a[i]) left++;
  while(a[right]>a[i]) right--;
  exchange(a[left++],a[right--]);
}while(left<right);

```

- Avoids separate arrays a_low, a_high (in-situ)
Pointers suffice, concat implicit
Cache friendly

Slide 7 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller




Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort IV

- Two scenarios for Parallelization:
 - data already in shared memory, processors all running
 - data must be read in, processors must be started seq.
- Latter: runtime $\Omega(n)$ speedup $O(\log n)$
ok for $p=O(\log n)$ i.e. small processor count
- Simple parallelization:
qsort(a_high) done on different processor if size $> n/p$
- Runtime: sequence of partitions $n+n/2+n/4+\dots=O(n)$
plus seq sorts $O(n/p \cdot \log(n/p))$

Slide 8 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller




Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort V

- Advanced: accelerate partition step!
- Approach 1: flatten hierarchy (Sample Sort)

choose $p-1$ pivots initially
each proc i partitions n/p elements from array a
into p partial lists ij according to pivots
each proc j gathers all partial lists ij into list j
each proc j sorts list j sequentially

Slide 9 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller




Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort VI

- Analysis:
 - partition time $O(n/p)$
 - seq sort $O(n/p \cdot \log(n/p))$
- Advantages:
 - no recursive calls
 - can also be used on message-passing machines
(one all-to-all communication)
- Disadvantage: not in-situ, lists ij need separate array

Slide 10 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller




Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort VII

- Approach 2: Tsigas' algorithm
- Keep divide-and-conquer, parallelize partition loop
- Each proc partitions part of array of size n/p
Then re-order partial partitions (details below)
- Partition processors into two sets
Choose number of processors for each partition in proportion to sizes of partition

Slide 11 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller




Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Quicksort VIII

- Partition done pagewise
- Page = block of constant size
- For each proc: ≤ 1 page with elements from both partit.
- Partition these pages sequentially in time $O(p)$
or parallel in time $O(\log p)$
- Re-order pages so that each partition in consecutive memory locations

Slide 12 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller



FerrariUniversität in Hagen Department of Mathematics and Computer Science

Quicksort IX

- Implementation: instead of left/right keep leftblock, rightblock
- Concurrent access to leftblock and rightblock managed either by lock or by fetch-and-add primitive

Slide 13 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort I

- No sequential counterpart!
- A sequence of numbers a_1, \dots, a_n is called *bitonic* if either there is a k such that $a_1 \leq \dots \leq a_k \geq \dots \geq a_n$ or the sequence can be rotated to that form
- Lemma (Batcher, 1968): If a is bitonic, then $a' = \min(a_1, a_{n/2+1}), \dots, \min(a_{n/2}, a_n)$ and $a'' = \max(a_1, a_{n/2+1}), \dots, \max(a_{n/2}, a_n)$ are both bitonic and $\max(a') \leq \min(a'')$
- Kind of divide rule for bitonic sequences

Slide 14 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort II

- Consequence of the Lemma:


```
sortb(int a[n],int which){ // a must be bitonic
compute a', a'' according to Lemma if which == asc
exchange max and min if which == desc
return(concat(sortb(a',which),sortb(a'',which))
}
```
- Analysis: bitonic seq can be sorted in time $O(\log n)$ with n proc.s
- Note: asc/desc order needed in a minute

Slide 15 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort III

- Turn arbitrary sequence into bitonic sequence by sorting its halves in ascending and descending order:


```
sort(int a[n],which){ // a is an arbitrary sequence
sort(a[1..n/2],asc); sort(a[n/2+1..n],desc); // now bitonic
sortb(a,which);
}
```
- Analysis for n proc.s: $T(n) = T(n/2) + O(\log n) = O((\log n)^2)$
- Not optimal but constant is very small

Slide 16 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort IV

- Example $n=8$

Slide 17 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort V

- Bitonic sort example of sorting network i.e. was intended for hardware
- In software: oblivious, i.e. control flow indep. of data
- With p processors: simple: each processor simulates n/p comparators
- better: stop recursion when size n/p is reached then sort sequentially

Slide 18 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort VI

- Call sequence rolls out in time into:

seqsort(n/p)	1	$O(n/p \cdot \log(n/p))$
sortb($2n/p$)	2	$O(n/p + n/p \cdot \log(n/p))$
sortb($4n/p$)	4	$O(2 \cdot n/p + n/p \cdot \log(n/p))$
...		
sortb($n/2$)	$p/2$	$O((\log p - 1) \cdot n/p + n/p \cdot \log(n/p))$
sortb(n)	p	$O(\log p \cdot n/p + n/p \cdot \log(n/p))$
- Parallel time $O(n/p \cdot \log p \cdot (\log p + \log(n/p)))$ on p proc.s

Slide 19 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort VII

- Oops, can we prove Batcher's Lemma?
- Either two-step proof: first prove Lemma for 0-1-sequences then construct mapping from arbitrary seq to 0-1-seq
- Direct proof: Restrict to bitonic sequence $a_1 \leq \dots \leq a_k \geq \dots \geq a_n$ Ok because rotating does not affect properties of a' a'' Restrict to $k \geq n/2$ Ok because otherwise consider sequence $a_n \dots a_1$.

Slide 20 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort VIII

- Restrict to case $a_{n/2} > a_n$ Otherwise $a_1 \dots a_{n/2}$ ascending $a_{n/2} \dots a_n$ bitonic
- There exists i with $k \leq i \leq n-1$ such that $a_{i-n/2} \leq a_i$ and $a_{i+1-n/2} > a_{i+1}$
- For $l = n/2 \dots i$: $\min(a_{i-n/2}, a_l) = a_{i-n/2}$
For $l = i+1 \dots n$: $\min(a_{i-n/2}, a_l) = a_l$
- Properties follow.

Slide 21 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Bitonic Sort IX

- Example

Slide 22 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Merge Sort I

- Merge: take two sorted blocks of length k combine into one sorted block of length $2k$
- Merge(int a[k], int b[k], int c[2k]){


```

int ap=0, bp=0, cp=0;
while(cp<2k){
  if(a[ap]<b[bp]) c[cp++] = a[ap++];
  else c[cp++] = b[bp++];
}

```
- Time: $O(k)$

Slide 23 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

FerrariUniversität in Hagen Department of Mathematics and Computer Science

Merge Sort II

- Idea: input data of length $n = n$ sorted blocks of length 1
- in round i , merge n/k sorted blocks of length $k=2^{i-1}$ into $n/(2k)$ sorted blocks of length $2k=2^i$
- After $\log n$ rounds: 1 sorted block of length n
- Analysis: Time $O(n \log n)$

Slide 24 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort III

- Widely used for external sort
Merge of blocks can be done by reading blocks pagewise
Is cache friendly!
- Preprocessing: don't start with blocks of size 1
For memory of size m
load data of size m
sort in mem
store blocks of size m
- Reduces number of rounds to $\log(n/m)$

Slide 25 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort IV

- Parallelization simple if $\geq 2p$ blocks available
Then p merges can work in parallel
- Analysis:
first $\log n - \log p$ rounds take time $O(n/p)$ each
last $\log p$ rounds take time $2n/p + 4n/p + \dots + n = O(n)$
- Simple parallel merge sort takes time $O(n + n/p * \log(n/p))$
on p proc.s
Optimal for $p \leq \log n$

Slide 26 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort V

- Improve rounds with $< 2p$ blocks: parallelize merge routine
- Merge with 2 processors:
split each input block a_i in two parts a_i^l, a_i^r such that
 $\text{length}(a_{\text{left}}^i) + \text{length}(a_{\text{right}}^i) = \text{length}(a_{\text{left}}^{i+1}) + \text{length}(a_{\text{right}}^{i+1})$
 $\max\{a_{\text{left}}^i, a_{\text{right}}^i\} \leq \min\{a_{\text{left}}^{i+1}, a_{\text{right}}^{i+1}\}$
- Then: $\text{merge}(a_{\text{left}}^i, a_{\text{right}}^i) = \text{concat}(\text{merge}(a_{\text{left}}^i, a_{\text{right}}^i), \text{merge}(a_{\text{left}}^{i+1}, a_{\text{right}}^{i+1}))$

Slide 27 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort VI

- Correctness guaranteed by 2nd property:
both proc.s can work independently
- Speedup comes from 1st property:
both proc.s have to work on half the data
- In practice: relax first property
allow slight imbalance, but reduce time to split blocks

Slide 28 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort VII

- Very simple method:
in block a_0 of length k , take elements at positions $k/2 - 2c, k/2 - c, k/2, k/2 + c, k/2 + 2c$
- For each element, find its position in block a_1 (bin. search)
- Take elements from a_1 and search their positions in a_0
- Split with the element giving best balance
- Time $O(\log k)$

Slide 29 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

Merge Sort VIII

- So far: dancehall parallelism
- On message-passing machines:
run all mergers of the merge tree concurrently
forward results pagewise
kind of tree-pipeline
Problem: distribution of mergers onto proc.s
- Further measures: SIMD parallelism in merge

Slide 30 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Summary I

- **Sorting algorithms are fascinating**
Parallel sorting algorithms are even more fascinating
- **Though rather old, still area of active research**
- **New architectures demand new or varied algorithms**
- **Principles mostly easy to grasp**
- **Engineering parallel sorting algorithms for performance is tedious and difficult**



Summary II

- **All algorithmic paradigms come into play**
- **In this lecture, only most common algorithms**
- **Many more: e.g. parallel rank sort**
- **So: stay tuned to news on sorting**



Summary III

- **Thanks a lot for your attention**

- **Questions?**

