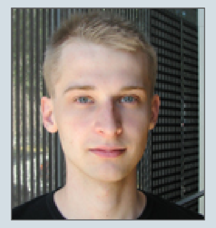
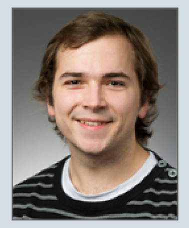
MADALGO CENTER FOR MASSIVE DATA ALGORITHMICS


Jakob Truelsen
Aarhus University


Mathias Rav
Aarhus University

Danmarks Grundforskningsfond Danish National Research Foundation

Adding Pipelining to TPIE

TPIE

TPIE (Templated Portable I/O Environment) is an external memory library with more than twenty years on the back. Developed at Duke University and Aarhus University, MADALGO. TPIE has many nice properties:

Efficiency
TPIE is written in templated C++ with a high focus on efficiency.

Ease of use
TPIE is easy to use by developers. It is well documented and not too invasive. TPIE is also easy to use for users – all you need to do is configure a dictionary for temporary files.

Resource management
TPIE keeps track of the memory and temporary space used. Resources such as memory are automatically allocated to the different part of programs.

Pipelining

The most common operation in an external memory algorithm is **sorting**. Often data is written to disk for the purpose of sorting, then a sorting function is invoked (which does a bunch of I/O's) and finally the result is read back from the disk.

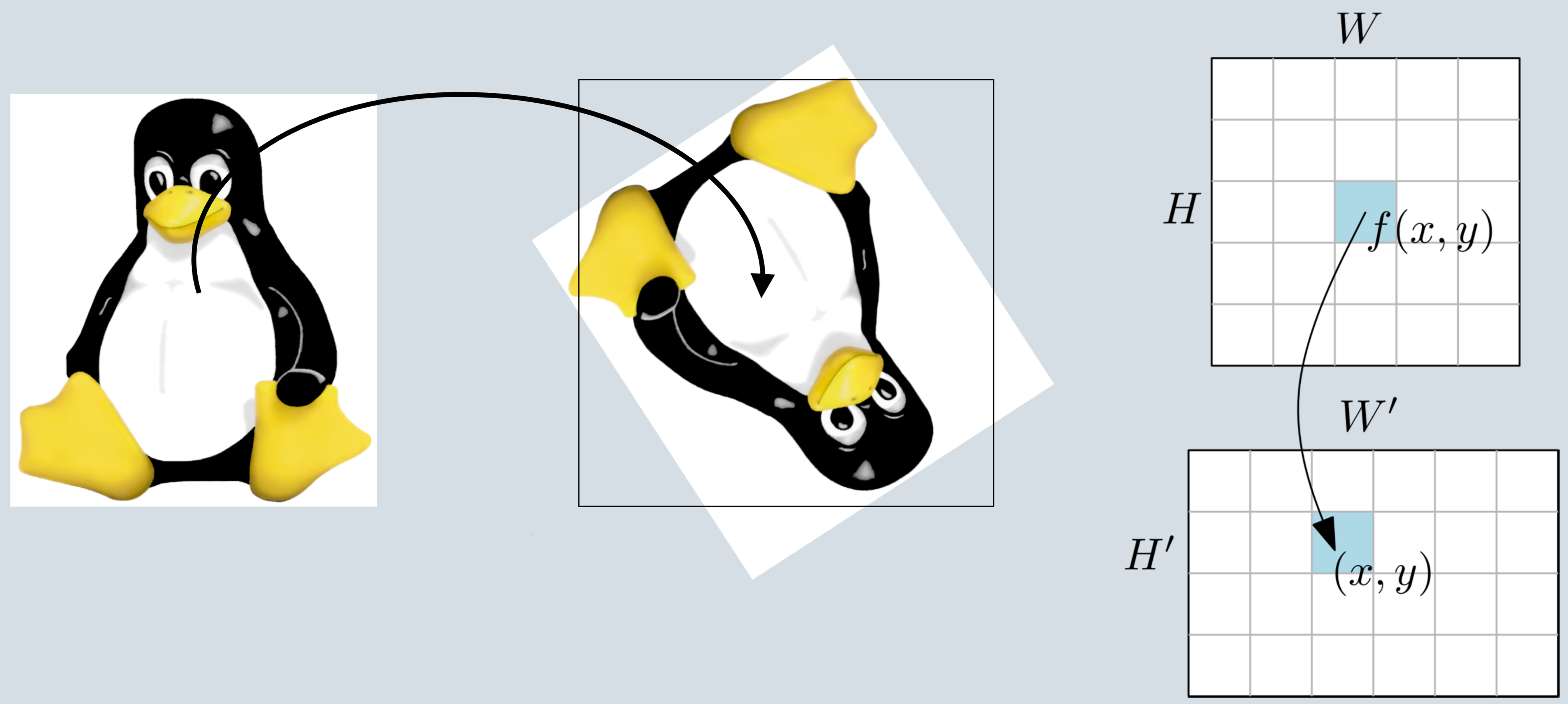
With **pipelining** we can save this writing and reading by feeding the elements to be sorted into a sorter and extracting them back out, typically saving a factor 2/3 in I/Os.

We have the following properties for pipelining:

- Better algorithm decomposition.
- Full compile time inlining and optimization.
- Automatically distribute memory for concurrent pipe nodes.
- Synchronous propagation of typed elements.
- Nice out of band data transfer.
- Automated phase scheduling.
- Automated progress tracking.
- Simple parallelism.
- Virtual chunks for dynamic runtime behaviour.

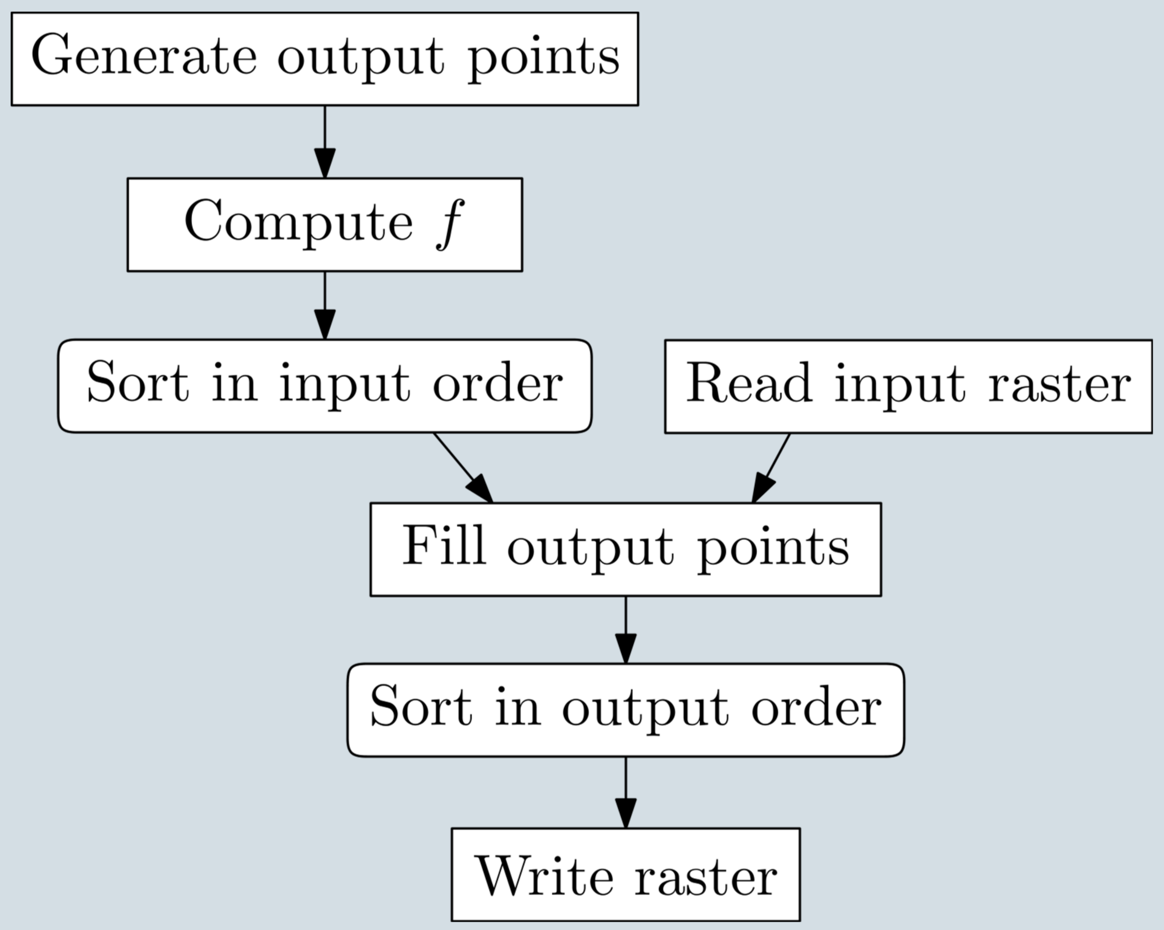
Example – Raster Mapping

In raster mapping an output image is created by applying a linear transformation to the coordinate space of an input raster, using a mapping *f*.



To not get holes in the output raster an inverse mapping is used, such that $O_{x,y} = I_{f^{-1}(x,y)}$.

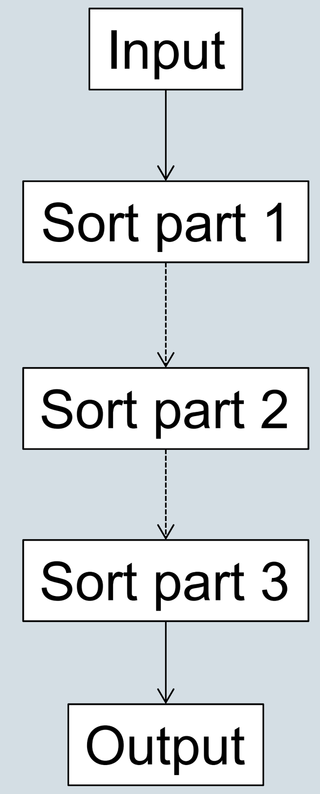
To perform this process in external memory, the following pipeline is used:



By using pipelined I/O two scans are saved for each sort, in practice more then halving the number of required I/Os.

Resource Management

To distribute memory each node in the graph is given a memory priority, a minimal amount of memory needed and a maximal amount of memory needed. Nodes are grouped in **phases** depending on what runs at the same time. The phases are run sequentially in some **topological order**.



In the example above “Input” and “Sort part 1” will run simultaneously in phase 1, “Sort part 2” by it self in phase 2 and “Sort part 3” and “Output” together in phase 3.

In phase 1 the “Input” node while have a priority of 0 and a min memory requirement of 2MiB while the “Sort part 1” will have a priority of 1. In this example we will allocate 2MiB to “Input” and the remaining memory to “Sort part 1”.

The general allocation strategy will ensure that:

- Every node is allocated the minimal amount of memory required (unless there is not enough memory).
- No node is allocated more then its max amount.
- All memory is allocated (unless this would violate max constraints).
- Memory is allocated as fairly as possible according to the priority, taking the max and min constrains into account.