Parallelizing a Tridiagonal System Solver by Adjustment to a Homomorphic Skeleton

Holger Bischof and Sergei Gorlatch

Technische Universität Berlin, Germany
{bischof,gorlatch}@cs.tu-berlin.de
DEVELOPMENT OF PARALLEL PROGRAMS NOWADAYS

- **Language**: mainly message passing, MPI
- **Design method**: ad hoc, based on invention
- **Performance**: hardly predictable during design
- **Correctness**: testing a few examples
- **Optimization**: by hand after implementation and profiling
- **Portability**: rather good because of the quasi standard MPI, but poor performance portability

Our approach: next slide
**SYSTEMATIC DESIGN WITH SKELETONS**

*Skeleton*: typical, recurring pattern of parallelism, customizable for a particular application.

- **Design method**: systematic *adjustment, customization*
- **Language**: e.g. MPI.
- **Correctness** of skeletons checked once in a formalism.
- **Optimizations**: provably correct, predictable impact
- **Performance**: predictable and portable, because only skeletons produce communication costs.
**SKELETONS: A FORMAL VIEW**

- *Skeleton*: higher order function on lists with a parallel implementation of good and predictable quality.
- BMF (Bird-Meertens Formalism): Functional formalism of higher order functions on lists with concatenation \( ++ \).
- Higher order functions are customizable by functions.

![Diagram of Skeletons and MPI program schema](image)
SKELETONS IN BMF

→ Elementary skeletons:

\[
\begin{align*}
map\ f\ [x_1, \ldots, x_n] & = [f\ x_1, \ldots, f\ x_n] \\
\text{scan}(\oplus)\ [x_1, \ldots, x_n] & = [x_1, x_1 \oplus x_2, \ldots, x_1 \oplus \cdots \oplus x_n] \\
\text{suffix}(\oplus)\ [x_1, \ldots, x_n] & = [x_1 \oplus \cdots \oplus x_n, \ldots, x_{n-1} \oplus x_n, x_n]
\end{align*}
\]

with parameters \( f, \oplus \) – maybe complex functions

→ \text{scan} and \text{suffix} are well-parallelizable if \( \oplus \) is associative.

directly implementable in MPI: \texttt{MPI\_Scan}.

→ Homomorphisms: powerful, well known and widely used

divide-and-conquer pattern:

\( h \) is a homomorphism for some operation \( \otimes \) if the following holds:

\[
\begin{align*}
\text{divide} & \quad \text{conquer} \\
h (x \oplus y) & = h x \otimes h y
\end{align*}
\]
**THE DH** *(DISTRIBUTABLE HOMOMORPHISM) SKELETON*

Performance lack of homomorphisms: combine operation \( \otimes \) is possibly not well parallelizable \( \implies \) DH as a special case.

The DH function \( h = (\oplus \uparrow \otimes) \) on lists is a homomorphism with special combine operation and parameters \( \oplus \) and \( \otimes \):

\[
\begin{align*}
\text{divide} & \\
\begin{array}{c}
\text{h(x ++ y)} \\
\text{= zip(\(\oplus\))(h x, h y) ++ zip(\(\otimes\))(h x, h y)}
\end{array}
\end{align*}
\]

where \( \text{zip(\(\otimes\))([x_1, \ldots, x_n], [y_1, \ldots, y_n]) = [x_1 \otimes y_1, \ldots, x_n \otimes y_n]} \)
TRIDIAGONAL SYSTEM SOLVER

Solution of the tridiagonal system of equations:

\[
\begin{pmatrix}
\vdots & \ddots & \ddots \\
\vdots & \ddots & \ddots \\
\vdots & \ddots & \ddots \\
\end{pmatrix}
\begin{pmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{pmatrix}
= \begin{pmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{pmatrix}
\]

Non zero elements on three diagonals, main, upper and lower.

Traditional algorithm: Gaussian elimination.

Literature: non-trivial parallelism, several algorithms suggested.

Case study: parallelization with skeletons:
1) specify with functions on lists.
2) adjust to a known skeleton (DH).
Goal: **List representation.**

\[
\begin{pmatrix}
0 & a_{12} & a_{13} \\
& a_{21} & a_{22} & a_{23} \\
& & \ddots & \ddots & \ddots \\
& & & a_{n-1,1} & a_{n-1,2} & a_{n-1,3} \\
& & & & a_{n,1} & a_{n,2} \\
& & & & & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\end{pmatrix}
= 
\begin{pmatrix}
a_{14} \\
a_{24} \\
\vdots \\
a_{n-1,4} \\
a_{n,4}
\end{pmatrix}
\]

Assume \( a_{11} = a_{n,3} = 0 \), so every row consists of exactly four values \( \rightarrow \) List of quadruples as data structure.

Goal: Gaussian elimination as **functions on lists** of quadruples.

- Eliminate lower diagonal using a binary operation \( ① \) from top to bottom (resp. left to right in list notation)
- Eliminate upper diagonal using a binary operation \( ② \) bottom-up (resp. right to left in list notation)

\[ tds = \text{suffix}(②) \circ \text{scan}(①) \], parameters \( ① \), \( ② \) see paper.

①, ② not assoc. \( \Rightarrow \) not directly parallelizable \( \Rightarrow \) towards DH.
The DH is a homomorphism with special combine operation, so the adjustment to DH starts with adjustment to homomorphism.

We have to find a combine operations $\ast$, so that:

$$tds \ (x \ast y) = (tds \ x) \ast (tds \ y)$$

Gaussian elimination on a matrix where all rows consist of three values unequal to zero:

Knowing the result of $tds \ x$ and $tds \ y$, we can...
ADJUSTMENT TO HOMOMORPHISM, CONT.

...look for the combine operation $\odot$.

\[ \begin{pmatrix}
    (a) & \circ & \circ & \circ \\
    (b) & \bullet & \circ & \circ \\
    (c) & \circ & \circ & \circ \\
\end{pmatrix} \rightarrow \begin{pmatrix}
    \bullet & \circ & \circ & \circ \\
    \circ & \circ & \circ & \circ \\
    \circ & \circ & \circ & \circ \\
\end{pmatrix} \]

$\rightarrow$ Rows (a) and (c) are generated by using the last row of the first $N$-matrix and the first row of the second $N$-matrix.

$\rightarrow tds (x ++ y) = (tds x) \odot (tds y) = map g_1 (tds x) ++ map g_2 (tds y)$

$g_1$ uses row (a) to eliminate the last column of the first $N$-matrix, $g_2$ uses row (c) to eliminate the first column of the second one.

$\rightarrow tds$ is a homomorphism, but DH has better performance, so ...
... we adjust the combine operation to fit the DH format.

- Rewrite RHS by expressing $\text{map}$ in terms of $\text{zip}$:
  \[
  \text{tds} (x \uplus y) = \text{zip}(g_1 \circ \pi_1)(\text{tds} x, \text{tds} y) \uplus \text{zip}(g_2 \circ \pi_2)(\text{tds} x, \text{tds} y),
  \]
  where $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$.

- Still not a DH-format: $g_1$ and $g_2$ depend on the last element of $x$ and the first element of $y$.

- Common solution: Complement $\text{tds}$ by auxiliary functions, so that the “tupled” function is a DH.

- $\text{tds}$ is “almost DH”: $\text{tds} = \text{map} \pi_1 \circ (p \uparrow q) \circ \text{map triple}$, where

\[
\begin{pmatrix}
  a_1 \\
  f_1 \\
  l_1
\end{pmatrix}
\begin{pmatrix}
  a_2 \\
  f_2 \\
  l_2
\end{pmatrix} =
\begin{pmatrix}
  a_1 \circ t_1 \\
  f_1 \circ t_1 \\
  t_2 \circ l_2
\end{pmatrix}
=
\begin{pmatrix}
  a_1 \\
  f_1 \\
  l_1
\end{pmatrix}
\begin{pmatrix}
  a_2 \\
  f_2 \\
  l_2
\end{pmatrix} =
\begin{pmatrix}
  t_2 \circ a_2 \\
  f_1 \circ t_1 \\
  t_2 \circ l_2
\end{pmatrix}
\]

$t_1 = l_1 \circ f_2$  \hspace{1cm}  $t_2 = l_1 \circ f_2$
A diagram illustrating the process of adjusting and customizing a skeletal program schema. The process involves:

1. Math specification
2. Composition of skeletons
3. Adjustment and customization
4. Implementation
5. Optimization

The diagram shows a systematic approach to adjusting and customizing the skeleton, leading to an automatically generated implementation. The process involves formally derived skeleton optimization and implementation.

Legend:
- Math
- Skeleton
- MPI program schema
- Adjustment + customization
- Formally derived implementation
RESULT OF DERIVATION – MPI PROGRAM

Make_triple(data, input_data);
for(d=0; d<k; d++) {
    neighbor=rank^(1<<d);
    MPI_Sendrecv(data, ..., neighbor, ..., recvbuf, ..., neighbor, ...);
    if(myrank<neighbor) data=p(data, recvbuf);
    else data=q(recvbuf, data);}
Take_first(output_data, data);

\[ \text{\textit{map \ triple}} \]
\[ \text{\textit{p} \leftrightarrow \textit{q}} \]
\[ \text{\textit{map } \pi_1} \]

\textit{p} and \textit{q} are parameters of the DH implementation

Communication flows in a hypercube-like fashion
**Performance of TDS**

- Time measurements on Cray T3E, 300MHz
- Sequential time calculated: one row operation \( \approx 1 \mu s \)
- \( 2^{17} \) elements / processor \( \Rightarrow \) growing problem size

![Cray T3E, 2^{17} elements/processor graph]

- Y-axis: time in sec
- X-axis: processes
- Red line: sequential
- Green line: TDS
**Performance of DH-implementation: Allreduce, Scan**

- Some collective operations of MPI are instances of DH skeleton
- Performance Comparison:
  - MPI native implementation vs. DH implementation
- Measured times with $2^{20}$ elements / processor
CONCLUSION

- Skeleton approach to develop parallel programs:
  Systematic adjustment to a composition of skeletons, after which the target implementation is obtained automatically.
- Communication and concurrency is hidden in skeletons.
- Case study: Adjusting \( tds \) to the DH skeleton.
- The DH skeleton can be implemented in MPI with good performance.