A pipelined parallel OSIC algorithm based on the square root Kalman Filter for heterogeneous networks

F. J. Martínez-Zaldívar\textsuperscript{1}  A. M. Vidal-Maciá\textsuperscript{2}  D. Giménez\textsuperscript{3}

\textsuperscript{1}Departamento de Comunicaciones  
Universidad Politécnica de Valencia (Spain)  
fjmartin@dcom.upv.es

\textsuperscript{2}Departamento de Sistemas Informáticos y Computación  
Universidad Politécnica de Valencia (Spain)  
avidal@dsic.upv.es

\textsuperscript{3}Departamento de Informática y Sistemas  
Universidad de Murcia (Spain)  
domingo@dif.um.es

HeteroPar ’07, Austin, Texas 2007
Introduction

MMSE-OSIC decoding procedure
The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

Parallel algorithm

Data decomposition
Processors tasks
Arithmetic cost and load balancing
Communications and scalability

Experimental results

Conclusions
Motivation

MIMO: Multiple Input Multiple Output systems

BLAST: Bell Labs Layered Space-Time Architecture

Use of multiple antennas in transmission/reception

- Aim: increase the capacity/reliability of the links
- Several architectures: D-BLAST, Turbo-BLAST, V-BLAST . . .

V-BLAST: Vertical-BLAST

Decoding alternatives:

- Maximum Likelihood: Sphere Decoding (SD), . . .
- Linear decoding (with polynomial complexity): Zero Forcing, MMSE (Minimum Mean Square Error), . . . with OSIC (Ordered Successive Interference Cancellation) versions

Our paper

Parallel algorithm for MMSE-OSIC

Potential applications

Multicarrier systems (OFDM in DVB-T), several thousands problem dimension
**Motivation**

**MIMO: Multiple Input Multiple Output systems**

**BLAST: Bell Labs Layered Space-Time Architecture**

Use of multiple antennas in transmission/reception

- **Aim:** increase the capacity/reliability of the links
- **Several architectures:** D-BLAST, Turbo-BLAST, V-BLAST . . .

**V-BLAST: Vertical-BLAST**

Decoding alternatives:

- **Maximum Likelihood:** Sphere Decoding (SD), . . .
- **Linear decoding (with polynomial complexity):** Zero Forcing, MMSE (Minimum Mean Square Error), . . . with OSIC (Ordered Successive Interference Cancellation) versions

**Our paper**

Parallel algorithm for MMSE-OSIC

**Potential applications**

Multicarrier systems (OFDM in DVB-T), several thousands problem dimension
Motivation

MIMO: Multiple Input Multiple Output systems

BLAST: Bell Labs Layered Space-Time Architecture

Use of multiple antennas in transmission/reception

- Aim: increase the capacity/reliability of the links
- Several architectures: D-BLAST, Turbo-BLAST, V-BLAST . . .

V-BLAST: Vertical-BLAST

Decoding alternatives:

- Maximum Likelihood: Sphere Decoding (SD), . . .
- Linear decoding (with polynomial complexity): Zero Forcing, MMSE (Minimum Mean Square Error), . . . with OSIC (Ordered Successive Interference Cancellation) versions

Our paper
Parallel algorithm for MMSE-OSIC
Potential applications
Multicarrier systems (OFDM in DVB-T), several thousands problem dimension
Motivation

MIMO: Multiple Input Multiple Output systems

BLAST: Bell Labs Layered Space-Time Architecture
Use of multiple antennas in transmission/reception
- Aim: increase the capacity/reliability of the links
- Several architectures: D-BLAST, Turbo-BLAST, V-BLAST ...

V-BLAST: Vertical-BLAST
Decoding alternatives:
- Maximum Likelihood: Sphere Decoding (SD), ...
- Linear decoding (with polynomial complexity): Zero Forcing, MMSE (Minimum Mean Square Error), ... with OSIC (Ordered Successive Interference Cancellation) versions

Our paper
Parallel algorithm for MMSE-OSIC

Potential applications
Multicarrier systems (OFDM in DVB-T), several thousands problem dimension
Motivation

MIMO: Multiple Input Multiple Output systems

BLAST: Bell Labs Layered Space-Time Architecture

Use of multiple antennas in transmission/reception

- Aim: increase the capacity/reliability of the links
- Several architectures: D-BLAST, Turbo-BLAST, V-BLAST...

V-BLAST: Vertical-BLAST

Decoding alternatives:

- Maximum Likelihood: Sphere Decoding (SD), ...
- Linear decoding (with polynomial complexity): Zero Forcing, MMSE (Minimum Mean Square Error),... with OSIC (Ordered Successive Interference Cancellation) versions

Our paper

Parallel algorithm for MMSE-OSIC

Potential applications

Multicarrier systems (OFDM in DVB-T), several thousands problem dimension
Outline

1. Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2. Parallel algorithm
   - Data decomposition
   - Processors tasks
   - Arithmetic cost and load balancing
   - Communications and scalability

3. Experimental results

4. Conclusions
MMSE-OSIC decoding

Target: solve \( y = Hx + v \)

- \( H = (h_1, h_2, \ldots, h_n) \in \mathbb{C}^{m \times n} \) known and full rank channel matrix
- \( x \): symbols to transmit (belong to a discrete symbol set)
- \( y \): observation vector
- \( v \): process noise

MMSE estimation

\[
\hat{x}_{\text{MMSE}} = \arg\min_{\hat{x}} \left\{ a^*(x - \hat{x})(x - \hat{x})^*a \right\}, \forall a
\]

where

\[
\hat{x}_{\text{MMSE}} = \left( \begin{array}{c} H \\ \sqrt{\alpha}I_n \end{array} \right)^\dagger \left( \begin{array}{c} y \\ 0 \end{array} \right) = H_a^\dagger y
\]

\[
\hat{x} = \left\lfloor \hat{x}_{\text{MMSE}} \right\rfloor = \left\lfloor H_a^\dagger y \right\rfloor
\]

OSIC: orderly estimation of \( x \) (\( \hat{x} \)), component by component (better performance)

- Estimation of the strongest component (highest signal-to-noise ratio)
- Deflation of the system: cancellation of the decoded component contribution to the received signal
- Repeat until all components are decoded

Martínez, Maciá and Giménez

A pipelined parallel OSIC algorithm
**MMSE-OSIC decoding**

**Target:** solve \( y = Hx + v \)

- \( H = (h_1, h_2, \ldots, h_n) \in \mathbb{C}^{m \times n} \) known and full rank channel matrix
- \( x \): symbols to transmit (belong to a discrete symbol set)
- \( y \): observation vector
- \( v \): process noise

**MMSE estimation**

\[
\hat{x}_{\text{MMSE}} = \text{arg min}_{\hat{x}} \{ a^*(\hat{x} - \hat{x})(\hat{x} - \hat{x})^* a \}, \forall a
\]

where

\[
\hat{x}_{\text{MMSE}} = \left( \begin{array}{c}
H \\
\sqrt{\alpha} I_n
\end{array} \right)^\dagger \left( \begin{array}{c}
y \\
0
\end{array} \right) = H_{\alpha}^\dagger y
\]

\[
\hat{x} = \left\lfloor \hat{x}_{\text{MMSE}} \right\rfloor = \left\lfloor H_{\alpha}^\dagger y \right\rfloor
\]

**OSIC:** orderly estimation of \( x \) (\( \hat{x} \)), component by component (better performance)

- Estimation of the strongest component (highest signal-to-noise ratio)
- Deflation of the system: cancellation of the decoded component contribution to the received signal
- Repeat until all components are decoded

**A pipelined parallel OSIC algorithm**
Introduction
Parallel algorithm
Experimental results
Conclusions

MMSE-OSIC decoding procedure
The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

**MMSE-OSIC decoding**

**Target:** solve $y = Hx + v$

- $H = (h_1, h_2, \ldots, h_n) \in \mathbb{C}^{m \times n}$ known and full rank *channel* matrix
- $x$: symbols to transmit (belong to a discrete symbol set)
- $y$: observation vector
- $v$: process noise

**MMSE estimation**

$$\hat{x}_{\text{MMSE}} = \text{arg min}_{\hat{x}} \{ a^* (x - \hat{x})(x - \hat{x})^* a \}, \forall a$$

$$= \left( \frac{H}{\sqrt{\alpha} I_n} \right)^\dagger \left( \begin{array}{c} y \\ 0 \end{array} \right) = H_{\alpha}^\dagger y$$

$$\hat{x} = \lfloor \hat{x}_{\text{MMSE}} \rceil = \lfloor H_{\alpha}^\dagger y \rceil$$

where

- $(\cdot)^\dagger$: Moore-Penrose pseudoinverse
- $\left( \frac{H}{\sqrt{\alpha} I_n} \right)$: *augmented* channel matrix
- $H_{\alpha}^\dagger$: first $m$ columns of the pseudoinverse of the *augmented* channel matrix. The rows are named *nulling vectors*.
- $\lfloor \cdot \rceil$: mapping on the symbol set. $(\cdot)^*$: conjugate transpose

**OSIC: orderly estimation of $x$ ($\hat{x}$), component by component (better performance)**

- Estimation of the strongest component (highest signal-to-noise ratio)
- Deflation of the system: cancellation of the decoded component contribution to the received signal
- Repeat until all components are decoded
Necessary data for $H^\dagger_\alpha$ alternative computation

$H^\dagger_\alpha$ can be computed as $H^\dagger_\alpha = P^{1/2}Q^*_\alpha$ instead of a pseudoinverse submatrix.

Necessary data:
- $P^{1/2}$ (lower triangular): square root factor of the solution error estimation covariance matrix.

\[
P = E\{(x - \hat{x})(x - \hat{x})^*\} = (\alpha I_n + H^*H)^{-1} = P^{1/2}P^{*/2}
\]

$(\alpha^{-1}$: signal-to-noise ratio)
- $Q_\alpha$: first $m$ rows of the augmented channel matrix $QL$-factorization unitary matrix $Q$:

\[
\begin{pmatrix}
H \\
\sqrt{\alpha}I_n
\end{pmatrix} = QL
\]

Use of the square root Kalman Filter iterations
- To compute $P^{1/2}$ and $Q_\alpha$ alternatively.
- Not necessary that whole channel matrix $H$ exists before beginning the computations.
- The algorithm processes $H$ row by row (or groups of $q$ consecutive rows) in every iteration.
Necessary data for $H_{\alpha}^+$ alternative computation

$H_{\alpha}^+$ can be computed as $H_{\alpha}^+ = P^{1/2}Q_{\alpha}^*$ instead of a pseudoinverse submatrix.

Necessary data:
- $P^{1/2}$ (lower triangular): square root factor of the solution error estimation covariance matrix

$P = E\{(x - \hat{x})(x - \hat{x})^*\} = (\alpha I_n + H^*H)^{-1} = P^{1/2}P^{*/2}$

$(\alpha^{-1}$: signal-to-noise ratio)
- $Q_{\alpha}$: first $m$ rows of the augmented channel matrix QL-factorization unitary matrix $Q: \begin{pmatrix} H \\ \sqrt{\alpha}I_n \end{pmatrix} = QL$

Use of the square root Kalman Filter iterations

- To compute $P^{1/2}$ and $Q_{\alpha}$ alternatively.
- Not necessary that whole channel matrix $H$ exists before beginning the computations
- The algorithm processes $H$ row by row (or groups of $q$ consecutive rows) in every iteration.
Outline

1. Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2. Parallel algorithm
   - Data decomposition
   - Processors tasks
   - Arithmetic cost and load balancing
   - Communications and scalability

3. Experimental results

4. Conclusions
**SRKF-OSIC algorithm**

**Input:** \( H = \left( H_0^*, H_1^*, \ldots, H_{m/q-1}^* \right) \), \( P^{1/2}_{(0)} = \frac{1}{\sqrt{\alpha}} I_n \) and \( Q_{\alpha,(0)} = 0 \)

**Output:** \( P^{1/2} = P^{1/2}_{(m/q)} \), \( Q_\alpha = Q_{\alpha,(m/q)} \)

---

**for** \( i = 0, \ldots, m/q - 1 \) **do**

Calculate \( \Theta(i) \) and apply in such a way that:

\[
E(i) \Theta(i) = \begin{pmatrix}
I_q & H_i P_{(i)}^{1/2} \\
0 & P_{(i)}^{1/2}
\end{pmatrix} \Theta(i) = \begin{pmatrix}
R_{e,(i)}^{1/2} & 0 \\
-K_{P,(i)} & P_{(i+1)}^{1/2}
\end{pmatrix} = F(i)
\]

**end for**

preserving the lower triangular structure of \( P_{(i)}^{1/2} \), \( P_{(i+1)}^{1/2} \) and \( I_q / R_{e,(i)}^{1/2} \).

\( \Gamma(i+1) = \left( 0_{I_q \times q}, 0_{I_q \times m-q(i+1) \times q} \right)^T \),

and \( Z(i) = -\left( \Gamma_{(i+1)}^* - H_i H_i^+ \right) \Gamma_{(i+1)}^* R_{e,(i)}^{-1/2} \)

---

**\( i \)th-iteration cost:** \( W_{\text{sec},i}(n, q) \)

\((4n + 3q + 6 + 6q(i + 1))q_n \) flops due to the multiplications \( H_i P_{(i)}^{1/2} \) and the Givens rotations applications.

**Total cost:** \( W_{\text{sec}}(m, n, q) \)

\( \approx 4n^2 m + 4nm^2 \) flops

At the end we get

\( P^{1/2} = P_{(m/q)}^{1/2} \)

\( Q_\alpha = Q_{\alpha,(m/q)} \)
**SRKF-OSIC algorithm**

**Input:** \( H = \left( H^e_0, H^e_1, \ldots, H^e_{m/q-1} \right)^*, P^{1/2}_{(0)} = \frac{1}{\sqrt{\alpha}} I_n \) and \( Q_{\alpha,(0)} = 0 \)

**Output:** \( P^{1/2} = P^{1/2}_{(m/q)}, Q_{\alpha} = Q_{\alpha,(m/q)} \)

**for** \( i = 0, \ldots, m/q - 1 \) **do**

Calculate \( \Theta_{(i)} \) and apply in such a way that:

\[
E_{(i)} \Theta_{(i)} = \begin{pmatrix}
I_q & H_i P^{1/2}_{(i)} \\
0 & P^{1/2}_{(i)} \\
-\Gamma_{(i+1)} & Q_{\alpha,(i)}
\end{pmatrix} \Theta_{(i)} = \begin{pmatrix}
R^{1/2}_{(i)} & 0 \\
K_{p,(i)} & P^{1/2}_{(i+1)} \\
Z_{(i)} & Q_{\alpha,(i+1)}
\end{pmatrix} = F_{(i)}
\]

**end for**

preserving the lower triangular structure of \( P^{1/2}_{(i)} / P^{1/2}_{(i+1)} \) and \( I_q / R^{1/2}_{e,(i)} \):

\[
\Gamma_{(i+1)} = \left(0^T_{I_q \times q}, I_q, 0^T_{(m-q(i+1)) \times q}\right)^T,
\]

and \( Z_{(i)} = -\left( \Gamma_{(i+1)} - H_i H^\dagger_{\alpha,(i+1)} \right)^* R^{-*/2}_{e,(i)} \)

**i\textsuperscript{th}-iteration cost:** \( W_{\text{sec},i}(n, q) \)

\( (4n + 3q + 6 + 6q(i + 1))qn \) flops due to the multiplications \( H_i P^{1/2}_{(i)} \) and the Givens rotations applications.

**Total cost:** \( W_{\text{sec}}(m, n, q) \)

\( \approx 4n^2 m + 4nm^2 \) flops

At the end we get

\[
p^{1/2} = p^{1/2}_{(m/q)}
\]

\[
Q_{\alpha} = Q_{\alpha,(m/q)}
\]
SRKF-OSIC algorithm

Input: \( H = \left( \begin{array}{c} H_0^e, H_1^e, \ldots, H_{m/q-1}^e \end{array} \right)^*, P_{1/2}^{(0)} = \frac{1}{\sqrt{\alpha}} I_n \) and \( Q_{\alpha(0)} = 0 \)
Output: \( P_{1/2} = P_{1/2}^{(m/q)}, Q_{\alpha} = Q_{\alpha(m/q)} \)

for \( i = 0, \ldots, m/q - 1 \) do

Calculate \( \Theta(i) \) and apply in such a way that:

\[
E(i) \Theta(i) = \begin{pmatrix}
I_q & H_i P_{1/2}^{(i)} \\
0 & P_{1/2}^{(i)} \\
-\Gamma(i+1) & Q_{\alpha(i)}
\end{pmatrix}
\]

\[
\Theta(i) = \begin{pmatrix}
R_{e(i)}^{1/2} & 0 \\
\bar{K}_{p(i)} & P_{1/2}^{(i+1)} \\
Z(i) & Q_{\alpha(i+1)}
\end{pmatrix} = F(i)
\]

end for

preserving the lower triangular structure of \( P_{1/2}^{(i)}, P_{1/2}^{(i+1)} \) and \( I_q / R_{e(i)}^{1/2} \),

\[
\Gamma(i+1) = \left( 0_{q \times q}^T, I_q, 0_{(m-q(i+1)) \times q}^T \right)^T
\]

and \( Z(i) = - \left( \Gamma(i+1) - H_i H_i^+ \alpha(i+1) \right)^* R_{e(i)}^{-1/2} \)

\( i^\text{th} \)-iteration cost: \( W_{sec,i}(n, q) \)

\( (4n + 3q + 6 + 6q(i + 1))q \) flops due to the multiplications \( H_i P_{1/2}^{(i)} \) and the Givens rotations applications.

Total cost: \( W_{sec}(m, n, q) \)

\approx 4n^2 m + 4nm^2 \) flops

At the end we get

\( P_{1/2}^{1/2} = P_{(m/q)}^{1/2} \)

\( Q_{\alpha} = Q_{\alpha(m/q)} \)
The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

SRKF-OSIC algorithm

Input: \( H = \left( H_0^e, H_1^e, \ldots, H_{m/q-1}^e \right)^* \), \( P_{1/2}^{(0)} = \frac{1}{\sqrt{\alpha}} I_n \) and \( Q_{\alpha,(0)} = 0 \)

Output: \( P_{1/2} = P_{1/2}^{(m/q)} \), \( Q_{\alpha} = Q_{\alpha,(m/q)} \)

\[
\text{for } i = 0, \ldots, m/q - 1 \text{ do} \\
\text{Calculate } \Theta_{(i)} \text{ and apply in such a way that:} \\
E_{(i)} \Theta_{(i)} = \begin{pmatrix} I_q & H_i P_{1/2}^{(i)} \\ 0 & P_{1/2}^{(i)} \\ -\Gamma_{(i+1)} & Q_{\alpha,(i)} \end{pmatrix} \\
\Theta_{(i)} = \begin{pmatrix} R_{e,(i)}^{1/2} & 0 \\ \bar{K}_{p,(i)} & P_{1/2}^{(i+1)} \\ Z_{(i)} & Q_{\alpha,(i+1)} \end{pmatrix} = F_{(i)} \\
\text{end for} \\
\]

preserving the lower triangular structure of \( P_{1/2}^{(i)} / P_{1/2}^{(i+1)} \) and \( I_q / R_{e,(i)}^{1/2} \),

\[
\Gamma_{(i+1)} = \left( 0_{T_{q \times q}}, I_q, 0_{T_{(m-q(i+1)) \times q}} \right)^T, \\
\text{and } Z_{(i)} = -\left( \Gamma_{(i+1)}^* - H_i H_i^\dagger \right)^* R_{e,(i)}^{-1/2} \\
\]

\( i \)-th iteration cost: \( W_{\text{sec},i}(n, q) \)

\( (4n + 3q + 6 + 6q(i+1))q \) flops due to the multiplications \( H_i P_{1/2}^{(i)} \) and the Givens rotations applications.

Total cost: \( W_{\text{sec}}(m, n, q) \)

\( \approx 4n^2 m + 4nm^2 \) flops

At the end we get

\[
\begin{align*}
P_{1/2} &= P_{1/2}^{(m/q)} \\
Q_{\alpha} &= Q_{\alpha,(m/q)}
\end{align*}
\]
SRKF-OSIC execution example

Example with $m = 6$, $n = 3$, $q = 2$

$i = 0$

\[
\begin{align*}
E(0)\Theta(0) &= F(0) \\
\begin{pmatrix}
I_2 & H_0 P_0^{1/2} \\
0 & P_{0}^{1/2}
\end{pmatrix} \Theta(0) &= \begin{pmatrix}
R_{e,0}^{1/2} & 0 \\
K_{p,0} & P_{1}^{1/2}
\end{pmatrix} \\
\begin{pmatrix}
1 & 0 & \times & \times & \times \\
0 & 1 & \times & \times & \times \\
0 & 0 & \times & 0 & 0 \\
0 & 0 & \times & \times & 0 \\
0 & 0 & \times & \times & \times \\
-1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} \Theta(0) &= \begin{pmatrix}
\times & 0 & 0 & 0 & 0 \\
\times & \times & 0 & 0 & 0 \\
\times & \times & \times & 0 & 0 \\
\times & \times & \times & \times & 0 \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{align*}
\]
**SRKF-OSIC execution example**

Example with $m = 6$, $n = 3$, $q = 2$

$i = 1$

\[
\begin{align*}
E_{(1)}\Theta_{(1)} &= F_{(1)} \\
\begin{pmatrix}
I_2 & H_1 P^{1/2}_{(1)} \\
0 & P^{1/2}_{(1)} \\
-\Gamma_{(2)} & Q_{\alpha,(1)}
\end{pmatrix} \begin{pmatrix}
\Theta_{(1)}
\end{pmatrix} &= \begin{pmatrix}
R^{1/2}_{e,(1)} & 0 \\
\bar{K}_{p,(1)} & P^{1/2}_{(2)} \\
Z_{(1)} & Q_{\alpha,(2)}
\end{pmatrix}
\end{align*}
\]

\[
\begin{pmatrix}
1 & 0 & x & x & x \\
0 & 1 & x & x & x \\
0 & 0 & x & 0 & 0 \\
0 & 0 & x & x & 0 \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & 0 & 0 \\
0 & 0 & x & 0 & 0 \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
0 & 0 & x & x & x \\
\end{pmatrix} = \begin{pmatrix}
x & 0 & 0 & 0 & 0 \\
x & x & 0 & 0 & 0 \\
x & x & x & 0 & 0 \\
x & x & x & x & 0 \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
Example with $m = 6$, $n = 3$, $q = 2$

$i = 2 = m/q - 1$. End!

$$
\begin{align*}
E(2) \Theta(2) &= \begin{pmatrix}
I_2 & H_2 P_{1/2}^{(2)} \\
0 & P_{1/2}^{(2)} \\
-\Gamma(3) & Q_{\alpha,(2)}
\end{pmatrix} \Theta(2) = \\
\begin{pmatrix}
1 & 0 & \times & \times & \times \\
0 & 1 & \times & \times & \times \\
0 & 0 & \times & \times & \times \\
0 & 0 & \times & \times & \times \\
0 & 0 & \times & \times & \times \\
0 & 0 & \times & \times & \times \\
-1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0
\end{pmatrix}
\end{align*}
$$

$$
P^{1/2} \iff P^{1/2}_{(3)}$$

$$
Q_{\alpha} \iff Q_{\alpha,(3)}$$
Outline

1. Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2. Parallel algorithm
   - Data decomposition
   - Processors tasks
   - Arithmetic cost and load balancing
   - Communications and scalability

3. Experimental results

4. Conclusions
Generic structure of initial and final matrices

Let $E(i)$ and $F(i)$ be defined as:

\[
E(i) = \begin{pmatrix}
I_q & H_i P^{1/2}(i) \\
0 & P^{1/2}(i)
\end{pmatrix}
\quad \Rightarrow \quad
(D(i) \mid C(i)) = F(i)
\]

\[
F(i) = \begin{pmatrix}
R_{e,(i)}^{1/2} & 0 \\
\overline{K}_{p,(i)} & P^{1/2}_{(i+1)}(i)
\end{pmatrix}
\]

Where $I_q$ is the identity matrix of appropriate size, $H_i$ is a matrix, $P^{1/2}(i)$ is the square root of $P(i)$, and $C(i)$ is a matrix. The notation $P_{(i+1)}(i)$ represents the processor subscript that owns the submatrix, and $n_j$ denotes a left superscript that denotes the number of rows of the submatrix.
Generic structure of initial and final matrices

\[
E(i) = \begin{pmatrix}
I_q & H_i P_{1/2}^i \\
0 & P_{1/2}^i \\
-\Gamma_{(i+1)} & Q_{\alpha,(i)}
\end{pmatrix}
\Rightarrow \left( D(i) \mid C(i) \right) \iff \left( \begin{array}{c}
R_{e,(i)}^{1/2} \\
K_{p,(i)} \\
Z_{(i)}
\end{array} \right) = F(i)
\]

Generic structure of \( D(i) \)

\[
(D(i))_{P_j} = \begin{pmatrix}
L(i) \\
M(i) \\
N(i)
\end{pmatrix}
= \begin{pmatrix}
L(i) \\
\vdots \\
n_0 [M(i)] \\
\vdots \\
n_{p-1} [M(i)] \\
N(i)
\end{pmatrix}
\]

with \( n = \sum_{j=0}^{p-1} n_j \)
Generic structure of initial and final matrices

\[ E(i) = \begin{pmatrix} \mathbf{I}_q & H_{ij}P_{i}^{1/2} & P_{i}^{1/2} & Q_{\alpha,(i)} \end{pmatrix} \Rightarrow (D(i) \mid C(i)) \leftarrow \begin{pmatrix} R_{e,(i)}^{1/2} & \mathbf{0} & \mathbf{0} & P_{(i+1)}^{1/2} \end{pmatrix} = F(i) \]

Generic structure of \( D(i) \)

\[ (D(i))_{P_j} = \begin{pmatrix} L_{(i)} \\ M_{(i)} \\ N_{(i)} \end{pmatrix} = \begin{pmatrix} L_{(i)}^{n_p-1} \mathbf{0} \\ \vdots \\ N_{(i)}^{n_0} \end{pmatrix} \]

with \( n = \sum_{j=0}^{p-1} n_j \)

Initial (\( D(i) \)) in \( P_0 \):

\[ (D(i))_{P_0} = \begin{pmatrix} L_{(i)} \\ M_{(i)} \\ N_{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_q \\ \mathbf{0} \\ -\Gamma_{(i+1)} \end{pmatrix} \]

Final (\( D(i) \)) in \( P_{p-1} \):

\[ (D(i))_{P_{p-1}} = \begin{pmatrix} L_{(i)} \\ M_{(i)} \\ N_{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{e,(i)}^{1/2} \\ \mathbf{K}_{p,(i)} \\ \mathbf{Z}(i) \end{pmatrix} \]
Generic structure of initial and final matrices

Generic structure\(^a\) of \(E_{(i)}\) and \(F_{(i)}\):

\[
E_{(i)} = \begin{pmatrix}
I_q & H_{(i)} P_{(i)}^{1/2} \\
0 & P_{(i)}^{1/2} \\
-\Gamma_{(i+1)} & Q_{\alpha,(i)}
\end{pmatrix} \Rightarrow \begin{pmatrix} D_{(i)} \mid C_{(i)} \end{pmatrix} \iff \begin{pmatrix} R_{e,(i)}^{1/2} \\
\overline{K}_{p,(i)} \\
0 \\
R_{(i+1)}^{1/2} \\
\bar{K}_{p,(i)} \\
Z_{(i)} \\
Q_{\alpha,(i+1)}
\end{pmatrix} = F_{(i)}
\]

Initial and final \(D_{(i)}\)

Initial \((D_{(i)})_{P_0}\):

\[
(D_{(i)})_{P_0} = \begin{pmatrix}
L_{(i)} \\
M_{(i)} \\
N_{(i)}
\end{pmatrix} = \begin{pmatrix}
I_q \\
0 \\
-\Gamma_{(i+1)}
\end{pmatrix}
\]

Final \((D_{(i)})_{P_{p-1}}\):

\[
(D_{(i)})_{P_{p-1}} = \begin{pmatrix}
L_{(i)} \\
M_{(i)} \\
N_{(i)}
\end{pmatrix} = \begin{pmatrix}
R_{e,(i)}^{1/2} \\
\overline{K}_{p,(i)} \\
Z_{(i)}
\end{pmatrix}
\]

\(^a\) denotes an entire matrix. \([\cdot]\) denotes part of the matrix. \((\cdot)_{P_j}/[\cdot]_{P_j}\): a processor subscript denotes the processor that owns the (sub)matrix. \(n_j[\cdot]\): a left superscript denotes the number of the rows of the submatrix.

\[\begin{align*}
D_{(i)} & \rightarrow P_j \\
\end{align*}\]

\[E_{(i)} = \begin{pmatrix}
I_q \\
0 \\
-\Gamma_{(i+1)}
\end{pmatrix} \rightarrow \begin{pmatrix} D_{(i)} \mid C_{(i)} \end{pmatrix} \iff \begin{pmatrix} R_{e,(i)}^{1/2} \\
\overline{K}_{p,(i)} \\
Z_{(i)} \\
Q_{\alpha,(i+1)}
\end{pmatrix} = F_{(i)}
\]

with \(n = \sum_{j=0}^{p-1} n_j\)
A pipelined parallel OSIC algorithm

Generic structure of initial and final matrices

Generic structure\(^a\) of \(E_i\) and \(F_i\): \(\left( \begin{array}{c|c} D_i & C_i \end{array} \right)\)

\(^a\)\(\cdot\) denotes an entire matrix. \(\cdot\) denotes part of the matrix. \((\cdot)_{P_j}/[\cdot]_{P_j}\): a processor subscript denotes the processor that owns the (sub)matrix. \(n_j[\cdot]\): a left superscript denotes the number of the rows of the submatrix.

\[
E_i = \begin{pmatrix}
 I_q & H_i P^{1/2}_{(i)} \\
 0 & P^{1/2}_{(i)}
\end{pmatrix}
\Rightarrow \left( \begin{array}{c|c} D_i & C_i \end{array} \right) \Leftarrow \begin{pmatrix}
 R^{1/2}_{e,i} & 0 \\
 \overline{K}_{p,i} & P^{1/2}_{(i+1)}
\end{pmatrix}
= F_i
\]

Generic structure of \(C_i\)

Their columns are distributed among the processors

\[
C_i = \left( [C_i]_{P_{p-1}}, \ldots, [C_i]_{P_j}, \ldots, [C_i]_{P_0} \right)
\]

\(P_j\) will own \(n_j\) columns of \(C_i\), with \(n = \sum_{j=0}^{P-1} n_j\)
Generic structure of initial and final matrices

Generic structure\(^a\) of \(E_{(i)}\) and \(F_{(i)}\):

\[
\begin{pmatrix}
I_q & H_p^{1/2} & P_p^{1/2} & Q_{\alpha_{(i)}}
\end{pmatrix} \Rightarrow \begin{pmatrix}
D_{(i)} \\ C_{(i)}
\end{pmatrix} \Leftarrow \begin{pmatrix}
R_e^{1/2} \\ K_p \\ Z(i) \\ Q_{\alpha_{(i+1)}}
\end{pmatrix} = F_{(i)}
\]

\(^a\) denotes an entire matrix. [\(\cdot\)] denotes part of the matrix. \((\cdot)_{P_j}[/\cdot]_{P_j}\): a processor subscript denotes the processor that owns the (sub)matrix. \(n_j[\cdot]\): a left superscript denotes the number of the rows of the submatrix.

Generic structure of \(C_{(i)}\)

Their columns are distributed among the processors

\[
C_{(i)} = \left[ C_{(i)} \right]_{P_{j-1}}, \ldots, \left[ C_{(i)} \right]_{P_j}, \ldots, \left[ C_{(i)} \right]_{P_0}
\]

\(P_j\) will own \(n_j\) columns of \(C_{(i)}\), with \(n = \sum_{j=0}^{P-1} n_j\)

Initial and final \(C_{(i)}\)

Initial \(C_{(i)}\):

\[
\begin{pmatrix}
H_p^{1/2} \\ P_p^{1/2} \\ Q_{\alpha_{(i)}}
\end{pmatrix}
\]

Final \(C_{(i)}\):

\[
\begin{pmatrix}
0 \\ P^{1/2}_{(i+1)} \\ Q_{\alpha_{(i+1)}}
\end{pmatrix}
\]
**Introduction**

**Parallel algorithm**

**Experimental results**

**Conclusions**

**Data decomposition**

**Processors tasks**

**Arithmetic cost and load balancing**

**Communications and scalability**

---

**Generic structure of initial and final matrices**

Generic structure\(^a\) of \(E_{(i)}\) and \(F_{(i)}\):

\[
E_{(i)} = \begin{pmatrix}
I_q & H_{(i)}^{1/2} \\
0 & P_{(i)}^{1/2} \\
-\Gamma_{(i+1)} & Q_{\alpha,(i)}
\end{pmatrix} \Rightarrow \begin{pmatrix} D_{(i)} \mid C_{(i)} \end{pmatrix} \Leftrightarrow \begin{pmatrix}
R_{e,(i)}^{1/2} \\
\overline{K}_{p,(i)} \\
Z_{(i)} \\
0 \\
P_{(i+1)}^{1/2}
\end{pmatrix} = F_{(i)}
\]

- \(a(\cdot)\) denotes an entire matrix. \([\cdot]\) denotes part of the matrix. \((\cdot)_{P_j} / [\cdot]_{P_j}\): a processor subscript denotes the processor that owns the (sub)matrix. \(n_j[\cdot]\): a left superscript denotes the number of the rows of the submatrix.

**Generic structure of \(C_{(i)}\)**

Their columns are distributed among the processors:

\[
C_{(i)} = \left( [C_{(i)}]_{P_{0-1}}, \ldots, [C_{(i)}]_{P_j}, \ldots, [C_{(i)}]_{P_0} \right)
\]

\(P_j\) will own \(n_j\) columns of \(C_{(i)}\), with \(n = \sum_{j=0}^{P-1} n_j\)

**Initial and final \(C_{(i)}\)**

Initial \(C_{(i)}\):

\[
\begin{pmatrix}
H_{(i)}^{1/2} \\
P_{(i)}^{1/2} \\
Q_{\alpha,(i)}
\end{pmatrix}
\]

Final \(C_{(i)}\):

\[
\begin{pmatrix}
0 \\
P_{(i+1)}^{1/2} \\
Q_{\alpha,(i+1)}
\end{pmatrix}
\]

---

Martínez, Maciá and Giménez

A pipelined parallel OSIC algorithm
Generic structure of initial and final matrices

**Generic structure** of $E(i)$ and $F(i)$:

$$E(i) = \left( \begin{array}{c|c} I_q & H_i P^{1/2}(i) \\ 0 & P^{1/2}(i) \\ -\Gamma_{i+1} & Q_{\alpha,(i)} \end{array} \right) \Rightarrow \left( D(i) \mid C(i) \right) \Leftarrow \left( \begin{array}{c|c} R^{1/2}_{e,(i)} & 0 \\ K_{p,(i)} & P^{1/2}_{(i+1)} \\ Z(i) & Q_{\alpha,(i+1)} \end{array} \right) = F(i)$$

**Initial data distribution (p = 3 processors)**

$$E(i) = \left( \begin{array}{c|c} D(i) & C(i) \end{array} \right) = \left( \begin{array}{c|c} (D(i))_{P_0} & [C(i)]_{P_2} \\ [C(i)]_{P_1} & [C(i)]_{P_0} \end{array} \right)$$

**Final (end of iteration) data distribution (p = 3 processors)**

$$F(i) = \left( \begin{array}{c|c} D(i) & C(i) \end{array} \right) = \left( \begin{array}{c|c} (D'(i))_{P_2} & [C'(i)]_{P_2} \\ [C'(i)]_{P_1} & [C'(i)]_{P_0} \end{array} \right)$$

Martínez, Maciá and Giménez  A pipelined parallel OSIC algorithm
Generic structure of initial and final matrices

**Generic structure** of $E(i)$ and $F(i)$:

$$E(i) = \begin{pmatrix} I_q & H_jP^{1/2}(i) \\ 0 & P^{1/2}(i) \end{pmatrix} \quad \Rightarrow \quad \left( D(i) \mid C(i) \right) \quad \Leftrightarrow \quad \left( \begin{array}{c} R^{1/2}(e,i) \\ \mathbf{K}(p,i) \\ Z(i) \end{array} \right) \quad \Rightarrow \quad \left( D(i) \mid C(i) \right) = F(i)$$

- $a(\cdot)$ denotes an entire matrix. $[\cdot]$ denotes part of the matrix. $P_j / [\cdot] P_j$: a processor subscript denotes the processor that owns the (sub)matrix. $n_j [\cdot]$: a left superscript denotes the number of the rows of the submatrix.

**Initial data distribution ($p = 3$ processors)**

$$E(i) = \begin{pmatrix} D(i) & C(i) \end{pmatrix} = \begin{pmatrix} D(i)_p & [C(i)]_p_2 & [C(i)]_p_1 & [C(i)]_p_0 \end{pmatrix}$$

**Final (end of iteration) data distribution ($p = 3$ processors)**

$$F(i) = \begin{pmatrix} D(i) & C(i) \end{pmatrix} = \begin{pmatrix} D'(i)_p & [C'(i)]_p_2 & [C'(i)]_p_1 & [C'(i)]_p_0 \end{pmatrix}$$
1 Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2 Parallel algorithm
   - Data decomposition
   - Processors tasks
     - Arithmetic cost and load balancing
     - Communications and scalability

3 Experimental results

4 Conclusions
Pipelined iteration (I): in $P_0$

In $P_0$

$$E'_i = E(i)\Theta(i,P_0)$$

$$= \begin{pmatrix} D(i) & C(i) & C(i) & C(i) \\ C(i) P_2 & C(i) P_1 & C(i) P_1 & C(i) P_1 \end{pmatrix} \Theta(i,P_0)$$

$$= \begin{pmatrix} I_2 & n_2 [0] & n_2 [0] & n_2 [0] \\ n_1 [0] & n_1 [0] & n_1 [0] & n_1 [0] \\ -\Gamma(i+1) & P_0 \end{pmatrix} \begin{pmatrix} H P_1^{1/2} \\ P_1^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix} \Theta(i,P_0)$$

$$= \begin{pmatrix} L(i) & n_2 [0] & n_2 [0] & n_2 [0] \\ n_1 [0] & n_1 [0] & n_1 [0] & n_1 [0] \\ n_0 [M(i)] & P_0 \end{pmatrix} \begin{pmatrix} C(i) P_2 \\ C(i) P_1 \\ C(i) P_1 \end{pmatrix} \begin{pmatrix} 0 \\ P_1^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix}$$

Remarks

In $P_0$ we must partially transform $E(i)$ by means of a unitary matrix $\Theta(i,P_0)$.
Pipelined iteration (I): in $P_0$

$E'(i) = E(i) \Theta(i),P_0$

$$= \begin{pmatrix} D(i)_{P_0} & C(i)_{P_2} & C(i)_{P_1} & C(i)_{P_0} \end{pmatrix} \Theta(i),P_0$$

$$= \begin{pmatrix} I \quad n_2 \quad 0 \quad n_1 \quad 0 \quad n_0 \quad 0 \quad -\Gamma(i+1) \end{pmatrix}_{P_0} \begin{pmatrix} C(i)_{P_2} & C(i)_{P_1} & 0 \end{pmatrix}_{P_0}$$

$$= \begin{pmatrix} L(i) \quad n_2 \quad 0 \quad n_1 \quad 0 \quad n_0 \quad M(i) \quad N(i) \end{pmatrix}_{P_0} \begin{pmatrix} C(i)_{P_2} & C(i)_{P_1} & 0 \end{pmatrix}_{P_0}$$

Remarks

$\Theta(i),P_0$ will only affect to data belonging to $P_0$
Pipelined iteration (I): in $P_0$

\[ E'_{(i)} = E_{(i)} \Theta_{(i),P_0} \]

\[ = \begin{pmatrix} (D_{(i)})_{P_0} & \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_2} & \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_1} & \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_0} \end{pmatrix} \Theta_{(i),P_0} \]

\[ = \begin{pmatrix} I_g & n_2 [0] & n_1 [0] & n_0 [0] & -\Gamma_{(i+1)} \end{pmatrix}_{P_0} \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_2} \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_1} \begin{bmatrix} H_{P}^{1/2} & P_{(i)}^{1/2} & Q_{\alpha,(i)} \end{bmatrix}_{P_0} \]

\[ = \begin{pmatrix} L_{(i)} & n_2 [0] & n_1 [0] & n_0 [M_{(i)}] & N_{(i)} \end{pmatrix}_{P_0} \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_2} \begin{bmatrix} C_{(i)} \end{bmatrix}_{P_1} \begin{bmatrix} 0 & P_{(i)}^{1/2} & Q_{\alpha,(i+1)} \end{bmatrix}_{P_0} \]

Remarks

This is the detailed structure of the data involved in this computation: $(D_{(i)})_{P_0}$ and $[C_{(i)}]_{P_0}$
Pipelined iteration (I): in $P_0$

\[ E'_i = E_i \Theta(i).P_0 \]
\[ = \begin{pmatrix} D(i)P_0 & [C(i)]P_2 & [C(i)]P_1 & [C(i)]P_0 \end{pmatrix} \Theta(i).P_0 \]
\[ = \begin{pmatrix} I_q & n_2 [0] & n_1 [0] & n_0 [0] & -\Gamma(i+1) \end{pmatrix} P_0 \]
\[ = \begin{pmatrix} L(i) & n_2 [0] & n_1 [0] & n_0 [M(i)] & N(i) \end{pmatrix} P_0 \]

Remarks

The target is the zeroing of $[H_i P^{1/2}(i)]P_0$ preserving the lower triangular structure of $P^{1/2}(i)/P^{1/2}(i+1)$ and $I_q/L(i)$

Martínez, Maciá and Giménez
A pipelined parallel OSIC algorithm
Pipelined iteration (I): in $P_0$

\[
E'_{(i)} = E_{(i)} \Theta_{(i),P_0}
\]
\[
= \begin{pmatrix}
D_{(i)} & C_{(i)} & C_{(i)} & C_{(i)} & P_0 \\
I & n_2 & n_1 & n_0 & 0 \\
0 & n_1 & n_0 & 0 & I_{(i+1)} \\
-\Gamma & 0 & 0 & 0 & 0
\end{pmatrix}
\[
= \begin{pmatrix}
L_{(i)} & C_{(i)} & C_{(i)} & C_{(i)} & 0 \\
L & n_2 & n_1 & n_0 & I_{(i+1)} \\
M & n_1 & n_0 & n_0 & 0 \\
N & 0 & 0 & 0 & 0
\end{pmatrix}
\]
\[
= \begin{pmatrix}
H & P^{1/2} & P^{1/2} & P^{1/2} & P_{(i+1)} \\
0 & P^{1/2} & Q_{\alpha,(i+1)} & Q_{\alpha,(i+1)} & 0
\end{pmatrix}
\]

Remarks

The result can be reused in the next iteration and in the same memory positions.
Pipelined iteration (I): in $P_0$

$$E'_{(i)} = E_{(i)} \Theta_{(i),P_0}$$

$$= \begin{pmatrix} (D_{(i)})_{P_0} & [C_{(i)}]_{P_2} & [C_{(i)}]_{P_1} & [C_{(i)}]_{P_0} \end{pmatrix} \Theta_{(i),P_0}$$

$$= \begin{pmatrix} I_d & n_2[0] & n_1[0] & n_0[0] \end{pmatrix}_{P_0} \begin{pmatrix} [C_{(i)}]_{P_2} & [C_{(i)}]_{P_1} & H_i P^{1/2}_{(i)} P^{1/2}_{(i)} & Q_{\alpha,(i)} \end{pmatrix}_{P_0}$$

$$= \begin{pmatrix} L_{(i)} & n_2[0] & n_1[0] & n_0[M_{(i)}] \end{pmatrix}_{P_0} \begin{pmatrix} [C_{(i)}]_{P_2} & [C_{(i)}]_{P_1} & 0 & P^{1/2}_{(i+1)} \end{pmatrix}_{P_0}$$

Remarks

Now $P_0$ must transfer the non-zero data of $(D_{(i)})_{P_0}$ to $P_1$
Pipelined iteration (I): in $P_0$

\[
E'_(i) = E_(i) \Theta(i).P_0 \\
= \left( \begin{array}{c} 
D(i) \end{array} \right)_{P_0} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_2} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_1} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_0} \right) \Theta(i).P_0 \\
= \left( \begin{array}{c} 
I_d \\
n_2 [0] \\
n_1 [0] \\
n_0 [0] \\
-\Gamma(i) 
\end{array} \right)_{P_0} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_2} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_1} \left[ \begin{array}{c} H_i P^{1/2} \\
P^{1/2} \\
Q^{\alpha,(i)} 
\end{array} \right]_{P_0} \right) \Theta(i).P_0 \\
= \left( \begin{array}{c} 
n_2 [0] \\
n_1 [0] \\
n_0 [M(i)] \\
n_0 [N(i)] 
\end{array} \right)_{P_0} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_2} \left[ \begin{array}{c} C(i) \end{array} \right]_{P_1} \left[ \begin{array}{c} 0 \\
P^{1/2} \\
Q^{\alpha,(i+1)} 
\end{array} \right]_{P_0} \right)
\]

Remarks

$P_0$ can start the $i+1$ iteration, provided that it has received $H_{i+1}$ from the data acquisition subsystem (pipelined behavior)
Pipelined iteration (II): in $P_1$

\[ E''_{(i)} = E'_{(i)} \Theta_{(i),P_1} \]
\[ = \begin{pmatrix} D_{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} C_{(i)} \end{pmatrix}_{P_2} \begin{pmatrix} C_{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} C_{(i)} \end{pmatrix}_{P_0} \end{pmatrix} \Theta_{(i),P_1} \]
\[ = \begin{pmatrix} L_{(i)} \\ n_2 \begin{bmatrix} 0 \end{bmatrix} \\ n_1 \begin{bmatrix} 0 \end{bmatrix} \\ n_0 \begin{bmatrix} M_{(i)} \end{bmatrix} \\ N_{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} C_{(i)} \end{pmatrix}_{P_2} \begin{pmatrix} H_{(i)}P_{(i)}^{1/2} \\ P_{(i)}^{1/2} \\ Q_{\alpha,(i)} \\ P_{(i+1)}^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix}_{P_1} \begin{pmatrix} 0 \\ P_{(i+1)}^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix}_{P_0} \]
\[ = \begin{pmatrix} L'_{(i)} \\ n_2 \begin{bmatrix} 0 \end{bmatrix} \\ n_1 \begin{bmatrix} M_{(i)} \end{bmatrix} \\ n_0 \begin{bmatrix} M_{(i)} \end{bmatrix}' \\ N'_{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} C_{(i)} \end{pmatrix}_{P_2} \begin{pmatrix} H_{(i)}P_{(i)}^{1/2} \\ P_{(i)}^{1/2} \\ Q_{\alpha,(i)} \\ P_{(i+1)}^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix}_{P_1} \begin{pmatrix} 0 \\ P_{(i+1)}^{1/2} \\ Q_{\alpha,(i+1)} \end{pmatrix}_{P_0} \]

Remarks

The same comments are applicable to $P_1$ and the data belonging to it.

Martínez, Maciá and Giménez  A pipelined parallel OSIC algorithm
Pipelined iteration (II): in $P_1$

\[
E''_i = E'_i \Theta(i)_{P_1}
\]
\[
= \begin{pmatrix}
(D(i))_{P_1} & [C(i)]_{P_2} & [C(i)]_{P_1} & [C(i)]_{P_0}
\end{pmatrix}
\Theta(i)_{P_1}
\]
\[
= \begin{pmatrix}
L(i) & n_2 [0] & n_1 [0] & n_0 [M(i)]
\end{pmatrix}
\begin{pmatrix}
[C(i)]_{P_2}
\end{pmatrix}
\begin{pmatrix}
H_iP_{1/2}^{(i)} & P_{i/2}^{(i)} & 0 & P_{1/2}^{(i+1)} & P_{1/2}^{(i+1)} & 0 & P_{1/2}^{(i+1)} & P_{1/2}^{(i+1)} & 0
\end{pmatrix}
\Theta(i)_{P_1}
\]
\[
= \begin{pmatrix}
L'(i) & n_2 [0] & n_1 [M(i)] & n_0 [M(i)]'
\end{pmatrix}
\begin{pmatrix}
[C(i)]_{P_2}
\end{pmatrix}
\begin{pmatrix}
0 & P_{1/2}^{(i+1)} & P_{1/2}^{(i+1)} & 0 & P_{1/2}^{(i+1)} & P_{1/2}^{(i+1)} & 0
\end{pmatrix}
\Theta(i)_{P_1}
\]

Remarks

At the end, $P_1$ must transfer the non-zero data of $(D(i))_{P_1}$ to $P_2$
Pipelined iteration (III): in $P_2$

In $P_2$

$$
E''''(i) = E''(i) \Theta(i), P_2
$$

$$
= \begin{pmatrix}
L'(i) \\
n_2 [0] \\
n_1 [M(i)]' \\
n_0 [M(i)]''
\end{pmatrix}_{P_2}

\begin{pmatrix}
H_i P^{1/2} \\
P^{1/2}(i) \\
Q_{\alpha(i)} \\
0
\end{pmatrix}_{P_2}

\begin{pmatrix}
0 \\
0 \\
0 \\
P^{1/2}(i+1) \\
Q_{\alpha(i+1)}
\end{pmatrix}_{P_0}
= \begin{pmatrix}
R^{1/2}_{e(i)} \\
K_{p(i)}' \\
Z
\end{pmatrix}_{P_0} = F(i)
$$

Remark

The same comments are applicable to $P_2$ and the data belonging to it.
Pipelined iteration (III): in $P_2$

In $P_2$

\[ E'''(i) = E''(i) \Theta(i),P_2 \]

\[ = \begin{pmatrix}
L'(i) \\
L''(i) \\
L'''(i)
\end{pmatrix}_{P_2}
\begin{pmatrix}
H_{i}P_{1/2}^{(i)} \\
0 \\
0
\end{pmatrix}_{P_2}
\begin{pmatrix}
L'_{(i)} \\
L''_{(i)} \\
L'''_{(i)}
\end{pmatrix}_{P_2}
\begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}_{P_2}
\begin{pmatrix}
P_1/2 \\
P_{(i+1)}^{1/2} \\
P_{(i+1)}^{1/2}
\end{pmatrix}_{P_1}
\begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}_{P_0}
\Theta(i),P_2
\]

\[ = \begin{pmatrix}
R_{e,(i)}^{1/2} \\
K_{p,(i)}^{1/2} \\
Z
\end{pmatrix}
\begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}
= F(i)
\]

**Remark**

At the end of the $i^{th}$ iteration, $P_{(i+1)}^{1/2}$ and $Q_{\alpha,(i+1)}$ are obtained, distributed by columns among the processors.
Pipelined iteration (III): in $P_2$

In $P_2$

\[
E'''_{(i)} = E''_{(i)} \Theta_{(i),P_2}
\]

\[
= \begin{pmatrix}
L'_{(i)} \\
{n_2[i]} \\
{n_1[M(i)]} \\
{n_0[M(i)]'} \\
N'_{(i)}
\end{pmatrix}_{P_2}
\begin{pmatrix}
H_{P_{1/2}}^{(i)} \\
P_{1/2}^{(i)} \\
Q_{\alpha,(i)} \\
P_{1/2}^{(i+1)} \\
0
\end{pmatrix}_{P_2}
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}_{P_0}
\Theta_{(i),P_2}
\]

\[
= \begin{pmatrix}
\begin{pmatrix}
L''_{(i)} \\
{n_2[i]} \\
{n_1[M(i)]} \\
{n_0[M(i)]''} \\
N''_{(i)}
\end{pmatrix}_{P_2}
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}_{P_2}
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}_{P_0}
\Theta_{(i),P_2}
\]

\[
= \begin{pmatrix}
R_{e,(i)}^{1/2} \\
K_{p,(i)}^{1/2} \\
Z \\
Q_{\alpha,(i+1)}
\end{pmatrix}
= F_{(i)}
\]

Remark

Hence, we get $E_{(i)} \Theta_{(i)} = E_{(i)} \Theta_{(i),P_0} \Theta_{(i),P_1} \Theta_{(i),P_2} = F_{(i)}$, in a pipelined way.
Pipeline snapshot

\[ H_{i+2} P_{i+2} P_0 \]

\[ H_{i+1} P_{i+1} P_1 \]

\[ H_i P_i P_2 \]

\[ L_i + 2 \rightarrow P_0 \]

It initializes \((D_{i+2})\) and gets \([C'_{i+2}]_{P_0}\)

\[ L_i + 1 \rightarrow P_1 \]

\[ L_i \rightarrow P_2 \]

\[ Z_{i+2} \rightarrow P_{p-1} \]

It gets \([C'_{i}]_{P_{p-1}}\)
Outline

1 Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2 Parallel algorithm
   - Data decomposition
   - Processors tasks
   - Arithmetic cost and load balancing
   - Communications and scalability

3 Experimental results

4 Conclusions
Parallel arithmetic cost

\[ E_{(i)} \Theta_{(i)} = \begin{pmatrix} I_q & 0 \\ -\Gamma_{(i+1)} & P_0 \end{pmatrix} P_0 \begin{pmatrix} H_i P_{(i)}^{1/2} & 0 \\ P_{(i)}^{1/2} & Q_{\alpha,(i)} \end{pmatrix} P_{P-1} \cdots \begin{pmatrix} H_i P_{(i)}^{1/2} & 0 \\ P_{(i)}^{1/2} & Q_{\alpha,(i)} \end{pmatrix} P_j \cdots \begin{pmatrix} H_i P_{(i)}^{1/2} & 0 \\ P_{(i)}^{1/2} & Q_{\alpha,(i)} \end{pmatrix} P_0 \Theta_{(i)} \]

Distributed matrix multiplication: \( \left[ H_i P_{(i)}^{1/2} \right]_{Pj} \)

- \( P_j \) needs \( H_i \) and \( P_{(i)}^{1/2} \) \( P_j \)
- The structure of \( P_{(i)}^{1/2} \) saves costs
- The index of the first column assigned to \( P_j \) will be named \( c_{0j} \)

Givens rotations applications

- The zeroes in the \( n_j \) columns of \( \left[ H_i P_{(i)}^{1/2} \right]_{Pj} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{(i)}^{1/2} / P_{(i+1)}^{1/2} \) and \( I_q/R_{e,(i)}^{1/2} \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

Cost
Parallel arithmetic cost

\[ E_{(i)} \Theta_{(i)} = \left( \begin{array}{c}
I_q \\
0 \\
-\Gamma_{(i+1)}
\end{array} \right) \begin{bmatrix}
P_{(i)}^{1/2} \\
Q_{\alpha,(i)}
\end{bmatrix} P_0 \begin{bmatrix}
P_{(i)}^{1/2} \\
Q_{\alpha,(i)}
\end{bmatrix} P_{p-1} \cdots \begin{bmatrix}
P_{(i)}^{1/2} \\
Q_{\alpha,(i)}
\end{bmatrix} P_{j} \cdots \begin{bmatrix}
P_{(i)}^{1/2} \\
Q_{\alpha,(i)}
\end{bmatrix} P_0 \right) \Theta_{(i)} = F_{(i)}

Distributed matrix multiplication: \( [H_i P_{(i)}^{1/2}]_{P_j} \)

- \( P_j \) needs \( H_i \) and \( [P_{(i)}^{1/2}]_{P_j} \)
- The structure of \( P_{(i)}^{1/2} \) saves costs
- The index of the first column assigned to \( P_j \) will be named \( c_{0j} \)

Givens rotations applications

- The zeroes in the \( n_j \) columns of \( [H_i P_{(i)}^{1/2}]_{P_j} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{(i)}^{1/2}/P_{(i+1)}^{1/2} \) and \( I_q/R_{e,(i)}^{1/2} \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

Cost

Martínez, Maciá and Giménez
A pipelined parallel OSIC algorithm
Parallel arithmetic cost

\[ E_{(i)} \Theta_{(i)} = \begin{pmatrix} I_q & H_{(i)} \frac{1}{2} P_{(i)} \frac{1}{2} & \cdots & H_{(i)} \frac{1}{2} P_{(i)} \frac{1}{2} \end{pmatrix} P_{p-1} \begin{pmatrix} 0 & P_{(i)} \frac{1}{2} \frac{1}{2} & \cdots & 0 \end{pmatrix} P_{p-1} \Theta_{(i)} = F_{(i)} \]

Distributed matrix multiplication: \[ \left[ H_{(i)} P_{(i)}^{1/2} \right]_{P_{j}} \]

- \( P_{j} \) needs \( H_{i} \) and \( P_{(i)}^{1/2} \)
- The structure of \( P_{(i)}^{1/2} \) saves costs
- The index of the first column assigned to \( P_{j} \) will be named \( c_{0j} \)

Givens rotations applications

- The zeroes in the \( n_{j} \) columns of \( \left[ H_{(i)} P_{(i)}^{1/2} \right]_{P_{j}} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{(i)}^{1/2} / P_{(i+1)}^{1/2} \) and \( I_q / R_{e,(i)}^{1/2} \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

Cost
Parallel arithmetic cost

\[ E_{(i)} \Theta_{(i)} = \begin{pmatrix} I_q & H_{j} \bar{P}_{1/2}^{(i)} & \cdots & H_{j} \bar{P}_{1/2}^{(i)} & \cdots & H_{j} \bar{P}_{1/2}^{(i)} \\ 0 & \bar{P}_{1/2}^{(i)} & \cdots & \bar{P}_{1/2}^{(i)} & \cdots & \bar{P}_{1/2}^{(i)} \\ -\Gamma_{(i+1)} & \bar{Q}_{\alpha,(i)} & \cdots & \bar{Q}_{\alpha,(i+1)} & \cdots & \bar{Q}_{\alpha,(i+1)} \\ P_0 & P_{p-1} & \cdots & P_{p-1} & \cdots & P_0 \end{pmatrix} = F_{(i)} \]

Distributed matrix multiplication: \( [H_{j} \bar{P}_{1/2}^{(i)}]_{P_j} \)

- \( P_j \) needs \( H_j \) and \( [P_{1/2}^{(i)}]_{P_j} \)
- The structure of \( P_{1/2}^{(i)} \) saves costs
- The index of the first column assigned to \( P_j \) will be named \( c_{0j} \)

Givens rotations applications

- The zeroes in the \( n_j \) columns of \( [H_{j} \bar{P}_{1/2}^{(i)}]_{P_j} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{1/2}^{(i)} / P_{1/2}^{(i+1)} \) and \( I_q / R_{1/2}^{(i)} e,(i) \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

Cost

Cost of the \( i^{th} \) iteration in \( P_j \): \( W_{P_j,i(n,q)} = (3q + 8n - 8c_{0j} - 4n_j + 14 + 6[i + 1]q)n_j \) flops
Parallel arithmetic cost

\[
E_{(i)} \Theta_{(i)} = \begin{pmatrix}
I_q & H_j P_{(i)}^{1/2} & \cdots & H_j P_{(i)}^{1/2} \\
0 & P_{(i)}^{1/2} & \cdots & P_{(i)}^{1/2} \\
-\Gamma_{(i+1)} & Q_{\alpha,(i)} & \cdots & Q_{\alpha,(i)} \\
\end{pmatrix}_{P_0} \begin{pmatrix}
P_0 \\
P_1 \\
P_{p-1} \\
P_{j} \\
P_{p-1} \\
P_1 \\
P_0 \\
\end{pmatrix} = F_{(i)}
\]

Distributed matrix multiplication: \( \left[ H_j P_{(i)}^{1/2} \right]_{P_j} \)

- \( P_j \) needs \( H_i \) and \( \left[ P_{(i)}^{1/2} \right]_{P_j} \)
- The structure of \( P_{(i)}^{1/2} \) saves costs
- The index of the first column assigned to \( P_j \) will be named \( c_{0j} \)

Givens rotations applications

- The zeroes in the \( n_j \) columns of \( \left[ H_j P_{(i)}^{1/2} \right]_{P_j} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{(i)}^{1/2} / P_{(i+1)}^{1/2} \) and \( I_q / R_{e,(i)}^{1/2} \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

Cost

Cost of all the iterations in \( P_j \): \( W_{P_j} (m, n, q) = \sum_{i=0}^{m/q-1} W_{P_j,i} (n, q) \)
Parallel arithmetic cost

\[ E_{(i)} \Theta_{(i)} = \begin{pmatrix} I_q & H_{i} P_{(i)}^{1/2} & \ldots & H_{i} P_{(i)}^{1/2} & \ldots \\ 0 & P_{(i)}^{1/2} & 0 & \ldots & 0 \\ -\Gamma_{(i+1)} & Q_{\alpha,(i)} & P_{(i)+1}^{1/2} & Q_{\alpha,(i+1)} & \ldots & 0 \\ P_{p-1} & 0 & P_{(i)+1}^{1/2} & Q_{\alpha,(i+1)} & \ldots & 0 \\ 0 & 0 & P_{p-1} & 0 & \ldots & 0 \\ \end{pmatrix}_{p_0} \begin{pmatrix} \Theta_{(i)} \\ \end{pmatrix} = F_{(i)} \]

### Distributed matrix multiplication: \( \begin{bmatrix} H_{i} P_{(i)}^{1/2} \end{bmatrix}_{P_j} \)

- \( P_j \) needs \( \Gamma_{(i+1)} \) and \( \begin{bmatrix} P_{(i)}^{1/2} \end{bmatrix}_{P_j} \)
- The structure of \( P_{(i)}^{1/2} \) saves costs
- The index of the first column assigned to \( P_j \) will be named \( c_0j \)

### Givens rotations applications

- The zeroes in the \( n_j \) columns of \( \begin{bmatrix} H_{i} P_{(i)}^{1/2} \end{bmatrix}_{P_j} \) are obtained column-wise and from right to left
- The lower triangular structure of \( P_{(i)}^{1/2} / P_{(i+1)}^{1/2} \) and \( I_q / R_{e,(i)}^{1/2} \) is preserved
- The structure of \( \Gamma_{(i+1)} \) and the initial value of \( Q_{\alpha,(0)} = 0 \) saves costs

### Cost

Null parallelization arithmetic overhead: \( W_{sec}(m, n, q) = \sum_{j=0}^{p-1} W_{p_j} \)
Load balancing criteria

Perfect load balance

- \( P_j \) in the \( i \)th iteration \( \Rightarrow P_k \) in the \( (i + j - k) \)th iteration
- Load balancing: equal iteration execution time in every processor:

\[
W_{P_j,i}(n,q)t_{w_j} = W_{P_k,i+j-k}(n,q)t_{w_k} \quad \forall j \neq k,
\]

\( (t_{w_j} \text{ and } t_{w_k} \text{ are the time per flop in } P_j \text{ and } P_k \text{ respectively}) \)

- Difficult or impossible to obtain \( n_j \) and \( n_k \), \( \forall j \neq k \) and \( \forall i \), satisfying \( \sum_{j=0}^{p-1} n_j = n \).

A relaxed load balance criterion

- A simpler load balancing criterion:

\[
W_{P_j,i}(n,q)t_{w_j} = W_{P_k,i}(n,q)t_{w_k} \quad \forall j \neq k
\]

- So ideally

\[
W_{P_j,i}(n,q)t_{w_j} = \frac{W_{seq,i}(n,q)t_{w_{sec}}}{S_{max}(p,n,q)} \quad \forall 0 \leq j \leq p - 1
\]

\( S_{max}(p,n,q) \) is the maximum speedup attainable in the heterogeneous parallel system.
Load balancing criteria

Perfect load balance

- $P_j$ in the $i$th iteration $\Rightarrow P_k$ in the $(i + j - k)$th iteration
- Load balancing: equal iteration execution time in every processor:

$$W_{P_j,i}(n,q)t_{wj} = W_{P_k,i+j-k}(n,q)t_{wk}, \forall j \neq k,$$

$(t_{wj}$ and $t_{wk}$ are the time per flop in $P_j$ and $P_k$ respectively)

- Difficult or impossible to obtain $n_j$ and $n_k, \forall j \neq k$ and $\forall i$, satisfying $\sum_{j=0}^{p-1} n_j = n$.

A relaxed load balance criterion

- A simpler load balancing criterion:

$$W_{P_j,i}(n,q)t_{wj} = W_{P_k,i+j-k}(n,q)t_{wk}, \forall j \neq k$$

- So ideally

$$W_{P_j,i}(n,q)t_{wj} = \frac{W_{seq,i}(n,q)t_{wsec}}{S_{max}(p,n,q)}, \forall 0 \leq j \leq p - 1$$

$S_{max}(p,n,q)$ is the maximum speedup attainable in the heterogeneous parallel system.
Load balancing criteria

### Perfect load balance

- $P_j$ in the $i$th iteration $\Rightarrow P_k$ in the $(i + j - k)$th iteration
- Load balancing: equal iteration execution time in every processor:

$$W_{P_j,i}(n,q)t_{wj} = W_{P_k,i+j-k}(n,q)t_{wk}, \forall j \neq k.$$  

($t_{wj}$ and $t_{wk}$ are the time per flop in $P_j$ and $P_k$ respectively)

- Difficult or impossible to obtain $n_j$ and $n_k$, $\forall j \neq k$ and $\forall i$, satisfying $\sum_{j=0}^{p-1} n_j = n$.

### A relaxed load balance criterion

- A simpler load balancing criterion:

$$W_{P_j,j}(n,q)t_{wj} = W_{P_k,j}(n,q)t_{wk}, \quad \forall j \neq k$$

- So ideally

$$W_{P_j,j}(n,q)t_{wj} = \frac{W_{\text{seq},i}(n,q)t_{w_{\text{sec}}}}{S_{\text{max}}(p,n,q)}, \quad \forall 0 \leq j \leq p - 1$$

$S_{\text{max}}(p,n,q)$ is the maximum speedup attainable in the heterogeneous parallel system.

Martínez, Maciá and Giménez  
A pipelined parallel OSIC algorithm
Load balancing criteria

Perfect load balance

- \( P_j \) in the \( i \)th iteration \( \Rightarrow P_k \) in the \((i + j - k)\)th iteration
- Load balancing: equal iteration execution time in every processor:

\[
W_{P_j,i}(n, q) t_{wj} = W_{P_k,i+j-k}(n, q) t_{wk} \quad \forall j \neq k,
\]

\((t_{wj} \text{ and } t_{wk} \text{ are the time per flop in } P_j \text{ and } P_k \text{ respectively})\)

- Difficult or impossible to obtain \( n_j \) and \( n_k \), \( \forall j \neq k \) and \( \forall i \), satisfying \( \sum_{j=0}^{p-1} n_j = n \).

A relaxed load balance criterion

- A simpler load balancing criterion:

\[
W_{P_j,i}(n, q) t_{wj} = W_{P_k,i}(n, q) t_{wk} , \quad \forall j \neq k
\]

- So ideally

\[
W_{P_j,i}(n, q) t_{wj} = \frac{W_{seq,i}(n, q) t_{wsec}}{S_{max}(p, n, q)} , \quad \forall 0 \leq j \leq p - 1
\]

\( S_{max}(p, n, q) \) is the maximum speedup attainable in the heterogeneous parallel system.
Load balancing criteria

Perfect load balance

- $P_j$ in the $i$th iteration $\Rightarrow P_k$ in the $(i + j - k)$th iteration
- Load balancing: equal iteration execution time in every processor:

$$W_{P_{j,i}(n,q)t_{w_j}} = W_{P_{k,i+j-k}(n,q)t_{w_k}}, \forall j \neq k,$$

$(t_{w_j}$ and $t_{w_k}$ are the time per flop in $P_j$ and $P_k$ respectively)

- Difficult or impossible to obtain $n_j$ and $n_k$, $\forall j \neq k$ and $\forall i$, satisfying $\sum_{j=0}^{p-1} n_j = n$.

A relaxed load balance criterion

- A simpler load balancing criterion:

$$W_{P_{j,i}(n,q)t_{w_j}} = W_{P_{k,i}(n,q)t_{w_k}}, \forall j \neq k$$

- So ideally

$$W_{P_{j,i}(n,q)t_{w_j}} = \frac{W_{seq,i(n,q)t_{w_{sec}}}}{S_{max}(p,n,q)}, \forall 0 \leq j \leq p - 1$$

$S_{max}(p,n,q)$ is the maximum speedup attainable in the heterogeneous parallel system.
The normalized relative speed of a processor and maximum speedup

**Definition:** $s_j$, the normalized relative speed of a processor $P_j$

$$s_j = \frac{1}{\sum_{r=0}^{p-1} \frac{t_{wr}}{t_{w_j}}}, \quad \forall \; 0 \leq j \leq p - 1$$

It verifies that

- $\sum_{j=0}^{p-1} s_j = 1$
- $t_{w_j} s_j = t_{w_k} s_k, \quad \forall \; j \neq k$
- if $P_j$ is $u$ times faster than $P_k$ then $s_j = us_k$

**The maximum speedup**

Let us suppose that the sequential algorithm is run in $P_f$ (the fastest processor of the heterogeneous network). Hence

$$S_{\text{max}}(p, n, q) = \frac{1}{s_f}$$
The normalized relative speed of a processor and maximum speedup

**Definition:** $s_j$, the normalized relative speed of a processor $P_j$

$$s_j = \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_j}}{t_{w_r}}}, \quad \forall \, 0 \leq j \leq p - 1$$

It verifies that

- $\sum_{j=0}^{p-1} s_j = 1$
- $t_{w_j} s_j = t_{w_k} s_k, \quad \forall \, j \neq k$
- If $P_j$ is $u$ times faster than $P_k$ then $s_j = u s_k$

**The maximum speedup**

Let us suppose that the sequential algorithm is run in $P_f$ (the fastest processor of the heterogeneous network). Hence

$$S_{\text{max}}(p, n, q) = \frac{1}{s_f}$$
Dynamic vs. static load balancing

Dynamic load balancing

Again, the relaxed load balancing criterion:

$$W_{p_j,i}(n, q) t_{wj} = \frac{W_{seq,i}(n, q) t_{wsec}}{S_{max}(p, n, q)}$$

$$= \frac{1}{s_f}, \quad \forall 0 \leq j \leq p - 1$$

$$(3q + 8n - 8c_0 - 4n - 14 + 6[i + 1]q) qn_j t_{wj} = \frac{(4n + 3q + 6 + 6q[i + 1]) qn t_{wf}}{s_f}, \quad \forall 0 \leq j \leq p - 1$$

$$= (4n + 3q + 6 + 6q[i + 1]) qn, \quad \forall 0 \leq j \leq p - 1$$

with $c_{0j} = 1$ and $c_{0j} = c_{0j-1} + n_{j-1}, \forall 1 \leq j \leq p - 1$.

The $n_j$ values can be obtained solving a second order equation, but they depend on the iteration index, $i$, so the load balance is dynamic.

Static load balancing

If we wish a static load balancing criterion, we can balance the workload for the worst case: $i = m/q - 1$
Dynamic vs. static load balancing

Dynamic load balancing

Again, the relaxed load balancing criterion:

\[
W_{P_j,i}(n, q)_{tw_j} = \frac{W_{seq,i}(n, q)_{tw_{sec}}}{S_{max}(p, n, q)}
\]

\[
= \frac{1}{s_f}, \quad 0 \leq j \leq p - 1
\]

(3q + 8n - 8c_0 - 4n_j + 14 + 6[i + 1]q)qn_j_{tw_j} = (4n + 3q + 6 + 6q[i + 1])qn_{tw_f}, \quad 0 \leq j \leq p - 1

(3q + 8n - 8c_0 - 4n_j + 14 + 6[i + 1]q)qn_j \frac{1}{s_j} = (4n + 3q + 6 + 6q[i + 1])qn, \quad 0 \leq j \leq p - 1

with \(c_{00} = 1\) and \(c_{0j} = c_{0j−1} + n_{j−1}, \forall 1 \leq j \leq p − 1\).

The \(n_j\) values can be obtained solving a second order equation, but they depend on the iteration index, \(i\), so the load balance is dynamic.

Static load balancing

If we wish a static load balancing criterion, we can balance the workload for the worst case: \(i = m/q - 1\)
Outline

1. Introduction
   - MMSE-OSIC decoding procedure
   - The square root Kalman Filter for MMSE-OSIC (SRKF-OSIC)

2. Parallel algorithm
   - Data decomposition
   - Processors tasks
   - Arithmetic cost and load balancing
   - Communications and scalability

3. Experimental results

4. Conclusions
Communications analysis (I)

**$P_j$-to-$P_{j+1}$ data to transfer**

- **$H_j$**: $qn$ elements
- **$L_{(i)}$** (lower triangular): $\frac{1}{2} q(q + 1)$ elements
- Nonzero elements of $M_{(i)}$: $q \sum_{k=0}^{j} n_k$ elements
- Nonzero elements of $Z_{(i)}$: $[i + 1]q$ elements

**Linear model for $P_j$-to-$P_{j+1}$ transfers**

Communication time for the $i^{th}$ iteration:

$$T_{C,P_j,i}(n, q) = \beta + \tau \left[ qn + \frac{1}{2} q(q + 1) + q \sum_{k=0}^{j} n_k + [i + 1]q \right]$$ seconds

where $\beta$ is the communication settling time and $\tau$ is the transfer time per element.
$P_j$-to-$P_{j+1}$ data to transfer

- $H_j$: $qn$ elements
- $L_{(i)}$ (lower triangular): $\frac{1}{2} q(q + 1)$ elements
- Nonzero elements of $M_{(i)}$: $q \sum_{k=0}^{j} n_k$ elements
- Nonzero elements of $Z_{(i)}$: $[i + 1]q$ elements

Linear model for $P_j$-to-$P_{j+1}$ transfers

Communication time for the $i^{th}$ iteration:

$$T_{C,P_j,i}(n, q) = \beta + \tau \left[ qn + \frac{1}{2} q(q + 1) + q \sum_{k=0}^{j} n_k + [i + 1]q \right] \text{ seconds}$$

where $\beta$ is the communication settling time and $\tau$ is the transfer time per element.
Communications analysis (II)

**Model A: simultaneous transfers**

Communication time for the $i$th iteration:

$$T_{C,i}^{(A)}(n, q, p) = \frac{m}{q-1} \sum_{i=0}^{m/q-1} T_{C,P_j,i}^{(A)}(n, q) = T_{C,P_{p-2}}^{(A)}(n, q)$$

If we ignore the pipeline filling or emptying time, the total communication time is:

$$T_{C}^{(A)}(m, n, q, p) = \Theta(mn) + \Theta\left(\frac{m^2}{q}\right)$$

**Model B: serial transfers**

Communication time for the $i$th iteration:

$$T_{C,i}^{(B)}(n, q, p) = \sum_{j=0}^{p-2} T_{C,P_j,i}^{(B)}(n, q)$$

If we ignore the pipeline filling or emptying time, the total communication time is:

$$T_{C}^{(B)}(m, n, q, p) = \Theta(mnp) + \Theta\left(\frac{m^2p}{q}\right)$$
Communications analysis (II)

Model A: simultaneous transfers

Communication time for the $i$th iteration:

$$T_{C,i}^{(A)}(n, q, p) = \max_{j=0,...,p-2}\{T_{C,Pj,i}(n, q)\} = T_{C,Pp-2}(n, q)$$

If we ignore the pipeline filling or emptying time, the total communication time is:

$$T_{C}^{(A)}(m, n, q, p) = \sum_{i=0}^{m/q-1} T_{C,i}^{(A)}(n, q, p) = \Theta(mn) + \Theta\left(\frac{m^2}{q}\right)$$

Model B: serial transfers

Communication time for the $i$th iteration:

$$T_{C,i}^{(B)}(n, q, p) = \sum_{j=0}^{p-2} T_{C,Pj,i}(n, q)$$

If we ignore the pipeline filling or emptying time, the total communication time is:

$$T_{C}^{(B)}(m, n, q, p) = \sum_{i=0}^{m/q-1} T_{C,i}^{(B)}(n, q, p) = \Theta(mnp) + \Theta\left(\frac{m^2p}{q}\right)$$
Isoefficiency scalability

Parallel overhead time

Only communication time: \( T_C^{(A/B)}(m, n, q, p) \)

Isoefficiency scalability

Total overhead time compared to sequential time:

\[
\begin{align*}
pt_C^{(A/B)}(m, n, q, p) & \approx T_{\text{sec}}(m, n, q) \\
\Theta(mn) + \Theta\left(\frac{m^2}{q}\right), \text{ model A} & \approx \Theta\left(n^2 m + \Theta(nm^2)\right) \\
\Theta(mnp) + \Theta\left(\frac{m^2p}{q}\right), \text{ model B} & = \Theta\left(n^2 m + \Theta(nm^2)\right)
\end{align*}
\]

Hence, if \( m = \Theta(n) \)

\[
m, n = \begin{cases} 
\Theta(p), & \text{model A} \\
\Theta(p^2), & \text{model B}
\end{cases}
\]

An interconnection network that implements model A: daisy chain. It is highly scalable and suited for heterogeneous systems.
Isoefficiency scalability

Parallel overhead time

Only communication time: $T_C^{(A/B)}(m, n, q, p)$

Isoefficiency scalability

Total overhead time compared to sequential time:

$$pT_C^{(A/B)}(m, n, q, p) \approx T_{\text{seq}}(m, n, q) \approx \Theta(n^2 m) + \Theta(nm^2)$$

$$\Theta(mn) + \Theta\left(\frac{m^2}{q}\right), \text{ model A}$$

$$\Theta(mnp) + \Theta\left(\frac{m^2 p}{q}\right), \text{ model B}$$

$$= \Theta(n^2 m) + \Theta(nm^2)$$

Hence, if $m = \Theta(n)$

$$m, n = \begin{cases} 
\Theta(p), \text{ model A} \\
\Theta(p^2), \text{ model B} 
\end{cases}$$

An interconnection network that implements model A: daisy chain. It is highly scalable and suited for heterogeneous systems.
System

Platform

- cc-NUMA architecture 1.3 GHz Itanium 2 multiprocessor (upto 16 processor available to one user).
- Hypercube organization: every node is made up of two sets of two processors.
- Communication bandwidth depends on the processors ubication.
- Programs coded in Fortran
- Proprietary MPI communications library

Heterogeneous behavior

- Repeating the operations (twice) in some processors (half processors)
- Heterogeneity is independent of problem size
- Maximum speedup: $3/4p$ (Maximum efficiency: 75%)
### System

<table>
<thead>
<tr>
<th><strong>Platform</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- cc-NUMA architecture 1.3 GHz Itanium 2 multiprocessor (upto 16 processor available to one user).</td>
</tr>
<tr>
<td>- Hypercube organization: every node is made up of two sets of two processors.</td>
</tr>
<tr>
<td>- Communication bandwidth depends on the processors ubication.</td>
</tr>
<tr>
<td>- Programs coded in Fortran</td>
</tr>
<tr>
<td>- Proprietary MPI communications library</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Heterogeneous behavior</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Repeating the operations (twice) in some processors (half processors)</td>
</tr>
<tr>
<td>- Heterogeneity is independent of problem size</td>
</tr>
<tr>
<td>- Maximum speedup: $3/4p$ (Maximum efficiency: 75%)</td>
</tr>
</tbody>
</table>
Load balance

Arithmetic time per iteration in every processor ($p = 16$, $m = n = 6000$ and $q = 20$)

Load balancing
Achieved in the last iterations as designed

Martínez, Maciá and Giménez
A pipelined parallel OSIC algorithm
Load balance

Arithmetic time per iteration in every processor ($p = 16$, $m = n = 6000$ and $q = 20$)

Load balancing
Achieved in the last iterations as designed
Efficiency

Parallel algorithm efficiency ($m = 6000$ and $q = 20$)

Loss of ideal efficiency (max. theoretical efficiency 75%) due to
- Unbalancing in the first iterations
- Communication time
Parallel algorithm efficiency ($m = 6000$ and $q = 20$)

Loss of ideal efficiency (max. theoretical efficiency 75%) due to:
- Unbalancing in the first iterations
- Communication time
Conclusions

Designed parallel algorithm

We designed a pipelined parallel algorithm for OSIC-MMSE problem, based on the square root Kalman Filter

Regular tasks

The tasks are regular so the load can be easily distributed according to the processor speed in a heterogeneous system

Static load balancing

The ideal load balancing is dynamic but the behavior of the proposed static load balancing was satisfactory with good efficiency results
Conclusions

**Designed parallel algorithm**

We designed a pipelined parallel algorithm for OSIC-MMSE problem, based on the square root Kalman Filter.

**Regular tasks**

The tasks are regular so the load can be easily distributed according to the processor speed in a heterogeneous system.

**Static load balancing**

The ideal load balancing is dynamic but the behavior of the proposed static load balancing was satisfactory with good efficiency results.
Conclusions

**Designed parallel algorithm**
We designed a pipelined parallel algorithm for OSIC-MMSE problem, based on the square root Kalman Filter.

**Regular tasks**
The tasks are regular so the load can be easily distributed according to the processor speed in a heterogeneous system.

**Static load balancing**
The ideal load balancing is dynamic but the behavior of the proposed static load balancing was satisfactory with good efficiency results.
Future work

- Hybrid implementation
  MPI+OpenMP implementation (good preliminary results in a SMP)

- Other architectures
  Implementation in real heterogeneous networks

- Dynamic load balancing
  Dynamic load balancing implementation and comparison with static load balancing

- Generalization
  Generalization for pipelining parallel processing
Future work

Hybrid implementation

MPI+OpenMP implementation (good preliminary results in a SMP)

Other architectures

Implementation in real heterogeneous networks

Dynamic load balancing

Dynamic load balancing implementation and comparison with static load balancing

Generalization

Generalization for pipelining parallel processing
Future work

Hybrid implementation
MPI+OpenMP implementation (good preliminary results in a SMP)

Other architectures
Implementation in real heterogeneous networks

Dynamic load balancing
Dynamic load balancing implementation and comparison with static load balancing

Generalization
Generalization for pipelining parallel processing
## Future work

<table>
<thead>
<tr>
<th>Hybrid implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI+OpenMP implementation (good preliminary results in a SMP)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation in real heterogeneous networks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic load balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic load balancing \ implementation and comparison with static load balancing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalization for pipelining parallel processing</td>
</tr>
<tr>
<td>Future work</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td><strong>Hybrid implementation</strong></td>
</tr>
<tr>
<td>MPI+OpenMP implementation (good preliminary results in a SMP)</td>
</tr>
<tr>
<td><strong>Other architectures</strong></td>
</tr>
<tr>
<td>Implementation in real heterogeneous networks</td>
</tr>
<tr>
<td><strong>Dynamic load balancing</strong></td>
</tr>
<tr>
<td>Dynamic load balancing implementation and comparison with static load balancing</td>
</tr>
<tr>
<td><strong>Generalization</strong></td>
</tr>
<tr>
<td>Generalization for pipelining parallel processing</td>
</tr>
</tbody>
</table>
Thank you very much for your attention

—oOo—

Questions?