Outline

1. Motivations and statement of the problem
2. A new implementation of NARROW BAND method
3. Numerical tests
Outline

1. Motivations and statement of the problem
2. A new implementation of NARROW BAND method
3. Numerical tests
Solving the following HJB equation for $n \geq 3$ on low-power low-memory computers

\[
\begin{aligned}
\left\{ \begin{array}{l}
\theta_t(t, x) + \max_{a \in A, \lambda \in \Lambda(x)} \{-\lambda f(x, a) \cdot \theta_x(t, x)\} = 0 & x \in \mathbb{R}^n, t \in [0, T] \\
\theta(0, x) = \chi_T(x) & x \in \mathbb{R}^n
\end{array} \right.
\end{aligned}
\]

\[
\Lambda(x) := \left\{ \begin{array}{ll}
\{1\} & x \notin T \\
[0, 1] & x \in T
\end{array} \right.
\]

Note that $\theta \in \{0, 1\}$ and $T(x) = \inf\{t : \theta(t, x) = 0\}$
The function $\theta$ 

$\theta(t, x) \in \{0, 1\}$
The starting point is the Narrow Band method (Sethian et al., 1995) for front propagation problems.
To avoid computation on all the grid at each time step we can store a dynamic list containing the node of the *narrow band*.

Where I compute? Compute

1. Fast
2. Difficulties in managing the NB
3. Work only in 1-2D!
Another problem

Accessing data from RAM memory in a ordered manner is much faster than access data in an almost random manner so the main problem for CPU time is to access data in the narrow band.

First attempts: usage of a sparse matrix and a quad-tree approach. CPU time not linear with respect the number of nodes in the front.

Ref.
Bokanowski, Zidani, ENUMATH 2005.
Outline

1. Motivations and statement of the problem

2. A new implementation of NARROW BAND method

3. Numerical tests
New idea

Compute the solution using only nodes stocked in the narrow band without using an additional matrix.

Main difficulties
1. Update the NB in short time
2. How to find neighbors N, E, S, W of nodes in the narrow band?

Remark
We recover the front only at the final time $T$. The intermediate values of the solution are lost if we do not write them during computation (maybe not so expensive).
New idea

Compute the solution using only nodes stocked in the narrow band without using an additional matrix.

Main difficulties

1. Update the NB in short time
2. How to find neighbors N,E,S,W of nodes in the narrow band?

Remark

We recover the front only at the final time $T$. The intermediate values of the solution are lost if we do not write them during computation (maybe not so expensive).
Motivations and statement of the problem
A new implementation of NARROW BAND method
Numerical tests

New idea

Compute the solution using only nodes stocked in the narrow band without using an additional matrix.

Main difficulties

1. Update the NB in short time
2. How to find neighbors N,E,S,W of nodes in the narrow band?

Remark

We recover the front only at the final time $T$. The intermediate values of the solution are lost if we do not write them during computation (maybe not so expensive).
First method proposed: SPARSE STATIC

We construct the following semi-dynamic structure which contains the nodes of the NB.

We stock only nodes we are interested for and we have a direct access to each line of the original matrix.
Second method proposed: SPARSE DYNAMIC

We stock only pointers which does not point to NULL.

In order to maintain the direct access to each line of the original matrix we store a pointer at the beginning of the previous line we are visiting.
Motivations and statement of the problem
A new implementation of NARROW BAND method
Numerical tests

UltraBee scheme (Depres, Lagoutiere (01), B., Z. (07))

1D linear advection

\[
\begin{aligned}
&\begin{cases}
    v_t + f(x)v_x = 0 & x \in \mathbb{R}, t \in [0, T] \\
    v(0, x) = v_0(x) & x \in \mathbb{R}^n
\end{cases} \\
V_i^{n+1} = V_i^n - \frac{\Delta t}{\Delta x} f(x_i)(V_{i+\frac{1}{2}}^{n,L} - V_{i-\frac{1}{2}}^{n,R})
\end{aligned}
\]

\[
V_{i+\frac{1}{2}}^{n,L} := \min(\max(V_i^n, b_i^+), B_i^+)
\]

If \(f(x_i) > 0\) \[
\begin{aligned}
&\begin{cases}
    b_i^+ := \max(V_i^n, V_{i-1}^n) + \frac{\Delta x}{\Delta t f(x_i)}(V_i^n - \max(V_i^n, V_{i-1}^n)) \\
    B_i^+ := \min(V_i^n, V_{i-1}^n) + \frac{\Delta x}{\Delta t f(x_i)}(V_i^n - \min(V_i^n, V_{i-1}^n))
\end{cases}
\end{aligned}
\]

O. Bokanowski, E. Cristiani, H. Zidani
An efficient implementation of NB methods for HJB eqs
**nD linear advection**

extension via the alternate direction method

**HJB equations**

\[
V_i^{n+1} = \min_{k=1, \ldots, N_\alpha} \left( V_i^{n+1, UB}(\alpha_k, \lambda) \right)
\]
Outline

1. Motivations and statement of the problem
2. A new implementation of NARROW BAND method
3. Numerical tests
We tested and compared our methods on the *minimum time problem*

Domain= \([-2, 2]^n\)
\[ T = B_n(0, \frac{1}{2}) \]
\[ f = (-1, -1, \ldots, -1) \]
\[ T = 0.5 \]
CFL=0.9
C++ on LINUX, optimized compilation with -O3 option
Results for 2D tests
Tests in 2D

PC PIV 256MB RAM 512KB cache, optimized compilation -O3

<table>
<thead>
<tr>
<th>nodes</th>
<th>FULL</th>
<th>SP-STATIC</th>
<th>SP-DYNAMIC</th>
<th>time steps</th>
<th>nodes in NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$50^2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>132</td>
</tr>
<tr>
<td>$100^2$</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
<td>14</td>
<td>272</td>
</tr>
<tr>
<td>$200^2$</td>
<td>0.35</td>
<td>0.06</td>
<td>0.05</td>
<td>28</td>
<td>532</td>
</tr>
<tr>
<td>$400^2$</td>
<td>2.86</td>
<td>0.24</td>
<td>0.21</td>
<td>56</td>
<td>1068</td>
</tr>
<tr>
<td>$800^2$</td>
<td>23.94</td>
<td>0.94</td>
<td>0.87</td>
<td>112</td>
<td>2124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nodes</th>
<th>FULL</th>
<th>SP-STATIC</th>
<th>SP-DYNAMIC</th>
<th>time steps</th>
<th>nodes in NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x</td>
<td>8.4x</td>
<td>3.9x</td>
<td>4.1x</td>
<td>2x</td>
<td>1.9x</td>
</tr>
</tbody>
</table>

O. Bokanowski, E. Cristiani, H. Zidani

An efficient implementation of NB methods for HJB eqs
Motivations and statement of the problem
A new implementation of NARROW BAND method
Numerical tests

Results for 3D tests
**Tests in 3D**

PC PIV 256MB RAM and AMD Opteron 8GB RAM 1MB cache

<table>
<thead>
<tr>
<th>nodes</th>
<th>FULL 256MB</th>
<th>FULL 8GB</th>
<th>SP-STATIC</th>
<th>SP-DYNAM</th>
<th>ts</th>
<th>NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$25^3$</td>
<td>0.02</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
<td>4</td>
<td>432</td>
</tr>
<tr>
<td>$50^3$</td>
<td>0.38</td>
<td>0.24</td>
<td>*0.18</td>
<td>0.18</td>
<td>7</td>
<td>2016</td>
</tr>
<tr>
<td>$100^3$</td>
<td>6.10</td>
<td>4.39</td>
<td>1.51</td>
<td>1.85</td>
<td>14</td>
<td>8016</td>
</tr>
<tr>
<td>$200^3$</td>
<td>$\times$</td>
<td>64.29</td>
<td>14.41</td>
<td>22.55</td>
<td>28</td>
<td>31416</td>
</tr>
<tr>
<td>$400^3$</td>
<td>$\times$</td>
<td>1175.00</td>
<td>146.78</td>
<td>343.25</td>
<td>56</td>
<td>125968</td>
</tr>
</tbody>
</table>

* 0.6 secs if $T = B_n(0, 1)$.

<table>
<thead>
<tr>
<th>nodes</th>
<th>FULL 256MB</th>
<th>FULL 8GB</th>
<th>SP-STATIC</th>
<th>SP-DYNAM</th>
<th>ts</th>
<th>NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>8x</td>
<td>$\infty$</td>
<td>$18.3x$</td>
<td>$10.2x$</td>
<td>$15.2x$</td>
<td>2x</td>
<td>4x</td>
</tr>
</tbody>
</table>
Tests in 4D

PC PIV 256MB RAM

<table>
<thead>
<tr>
<th>total nodes</th>
<th>SP-STATIC</th>
<th>time steps</th>
<th>nodes in NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$25^4$</td>
<td>0.49</td>
<td>4</td>
<td>3024</td>
</tr>
<tr>
<td>$50^4$</td>
<td>*6.39</td>
<td>7</td>
<td>23552</td>
</tr>
<tr>
<td>$100^4$</td>
<td>123.97</td>
<td>14</td>
<td>179472</td>
</tr>
</tbody>
</table>

* 7.37 secs for 2 controls
* 9.04 secs for 4 controls.

<table>
<thead>
<tr>
<th>total nodes</th>
<th>SP-STATIC</th>
<th>time steps</th>
<th>nodes in NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x</td>
<td>19.4x</td>
<td>2x</td>
<td>7.6x</td>
</tr>
</tbody>
</table>
The SPARSE-STATIC method seems to be the best

- Very easy to implement
- CPU time is at least 1/10 wrt the FULL method if $N$ is large enough
- Efficient use of the memory. It is possible to run algorithms in 4D!