



Distributed Object-Oriented Data Structures and Algorithms for VLSI CAD

John A. Chandy

Cadence Design Systems

Steven Parkes

Sierra Vista Research

Prithviraj Banerjee

University of Illinois

Third International Workshop on Parallel Algorithms for
Irregularly Structured Problems

August 20 1996



Outline

- **Motivations**
- **ProperCAD II**
- **Example application - parallel standard cell placement**
- **Conclusions**

Motivations

- Existing algorithms for VLSI CAD applications running on uniprocessors inadequate for future requirements
- Parallel processing offers more performance for reduced design turnaround time and larger problem sizes or better quality solutions
- Parallel processing affordable: high speed network of workstations, multiprocessor workstations
- VLSI CAD algorithms are characterized by irregular data structures
- Need a model to effectively design parallel applications

ProperCAD Project

Applications

<i>ProperEXT</i>	<i>Extraction</i>
<i>ProperDRC</i>	<i>Layout Verification</i>
<i>ProperTEST</i>	<i>ATPG</i>
<i>ProperGA</i>	<i>ATPG</i>
<i>ProperSYN</i>	<i>Synthesis</i>
<i>ProperMIS</i>	<i>Synthesis</i>
<i>ProperPLACE</i>	<i>Placement</i>
<i>ProperROUTE</i>	<i>Routing</i>
<i>ProperHITEC</i>	<i>ATPG</i>
<i>ProperPROOFS</i>	<i>Fault Simulation</i>
<i>ProperSIM</i>	<i>Circuit Simulation</i>
<i>ProperVHDL</i>	<i>VHDL Simulation</i>

Existing Serial Algorithms

Parallel Application

TimberWolfSC
HITEC/PROOFS
MIS/SIS
..

ProperCAD Library

Actor Interface

Abstract Parallel Architecture

Multicomputers

Multiprocessors

Hybrids

Intel iPSC
Intel Paragon
IBM SP-1
TMC CM-5

SUN MP
Encore
Sequent
SGI

Workstation clusters

IRREGULAR '96

4

Actors

- **Based on work by Agha, Hewitt**
- **Fundamental object called actor provides concurrency mechanism**
- **Actor consists of a thread of control that communicates with other actors with messages**
- **All actor actions are in response to a message**
- **Three basic actions**
 - **create a new actor**
 - **send a message**
 - **change its behavior**

Actor model with CPS

ProperCAD II interface

Actor creation through inheritance

```
class Foo : public Actor {...};
```

Actor names serve the equivalent of pointers valid across all address spaces

```
Foo *actorPtr = ...;  
ActorName<Foo> fooName = actorPtr;
```

Actor methods allow the execution of concurrent methods

```
class Foo : public Actor {  
    void bar( barArgs& );  
    class bar : public ActorMethod<barArgs> {};  
}
```

```
barArgs &args;  
Foo::bar::Continuation cont( fooName );  
cont( args );
```

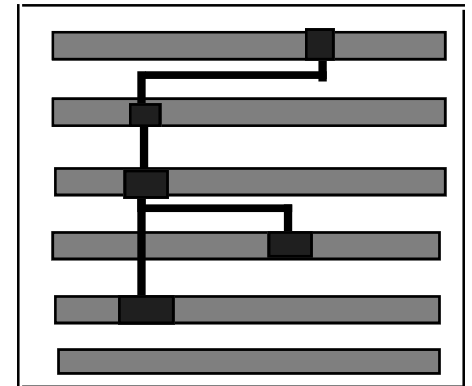
ProperCAD II features

- **Support for aggregates (Chien)**
- **Allows modification of run time policies**
 - queuing strategies
 - load balancing
 - memory usage
- **Portable across many architectures because of APA**

Standard cell placement

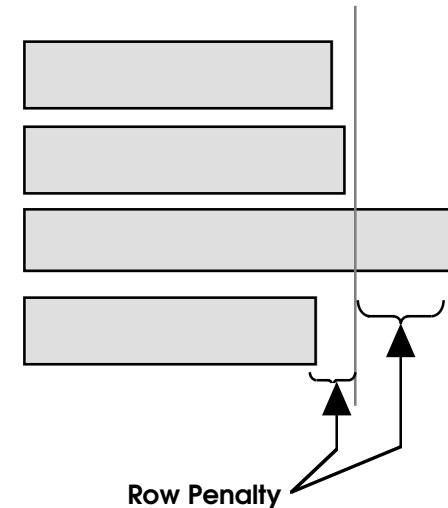
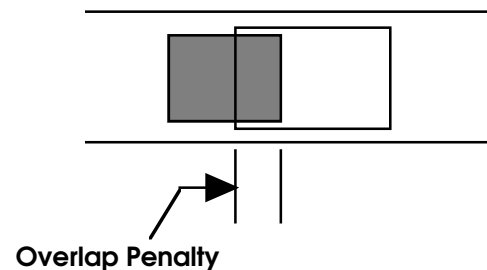
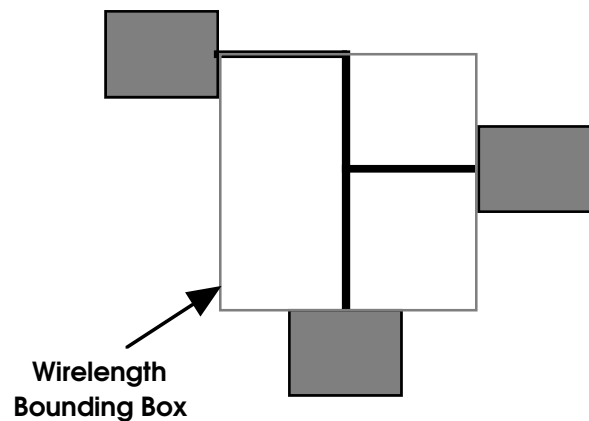
- Given cells and connections, place them to minimize wirelength and area
- Simulated annealing is most common algorithm

```
Initial_place_cells;
for(T=T_initial; T < T_final; T = a . T) {
  for(i=0;i<max_moves; i++) {
    Select two random cells;
     $\Delta$  = Evaluate_cost_change(exchange positions);
    Accept if negative, or with prob =  $e^{-\{\Delta / T\}}$ 
    Update state;
  }
}
```



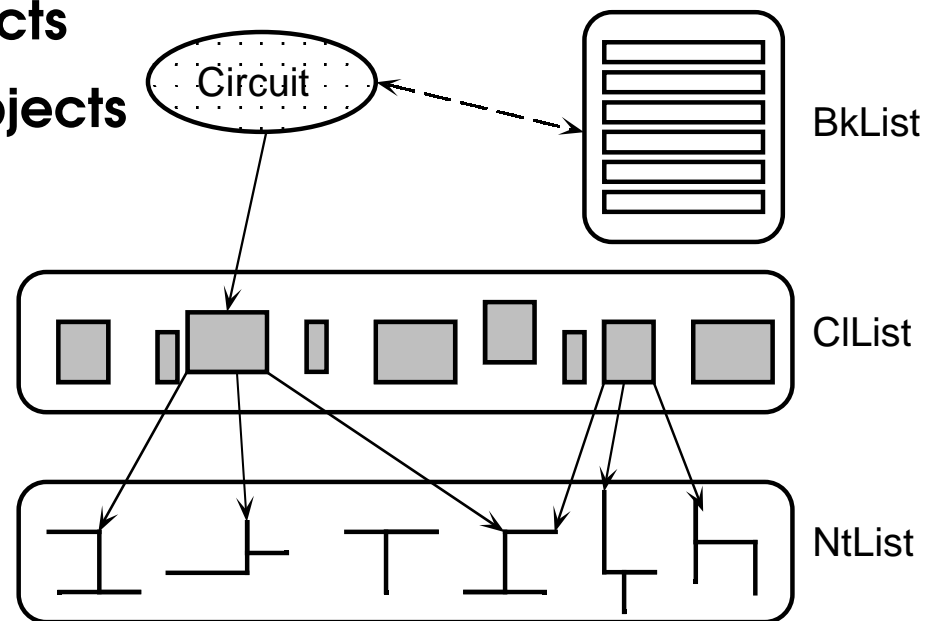
TimberWolfSC

- **TimberWolfSC 6.0 (Sechen)** is one of the leading sequential applications
 - cost function $C = W + \mu P_o + IP_R$
 - range limiter, incremental wirelength calculation, early rejection, feedback control



Object-Oriented Cell Placement

- Based on TimberWolfSC 6.0
- C++ classes
 - CList - collection of Cell objects
 - NtList - collection of Net objects
 - BkList - collection of Block objects
 - Circuit - manager



Serial code

```
class Circuit {
    ClList &carray;
    NtList &netarray;
    BkList &barray;
    // ...
};

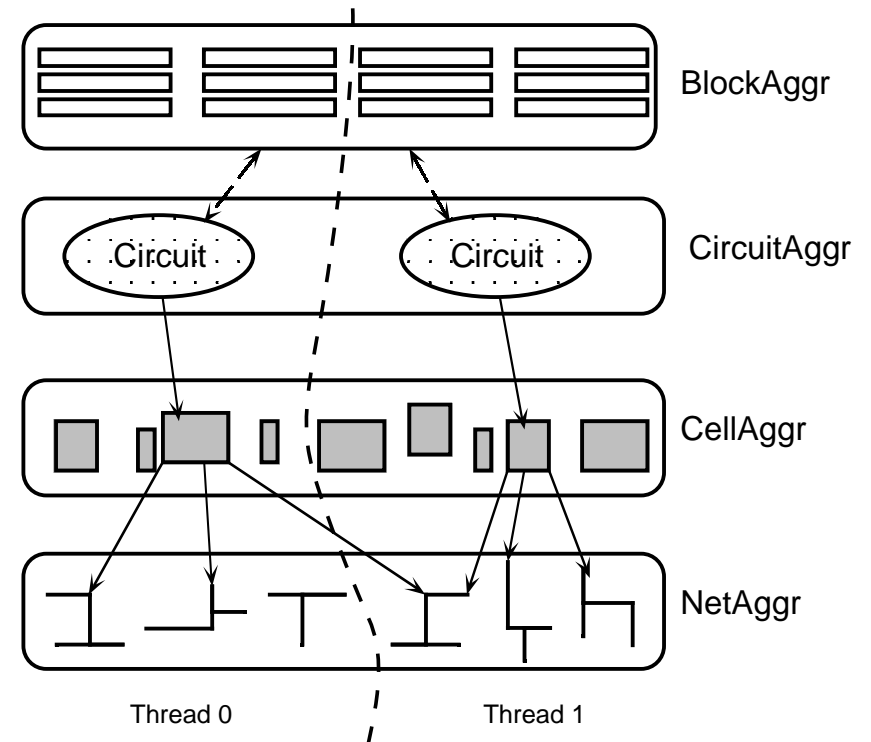
Circuit::anneal()
{
    while ( terminationNotReached() )
        tryCellMove();
    finishUp();
}
```

Serial code

```
Circuit::tryCellMove()  
{  
    Cell &cell = carray.pickACell();  
    newLocation = barray.newLocation();  
    deltaCost = carray.evaluateMove( cell, newLocation );  
    if ( acceptMove( deltaCost ) )  
        carray.updateCell( cell, newLocation );  
}  
  
CList::updateCell( Cell &cell, Position &newLocation )  
{  
    for ( pinIterator pin(cell); pin.end(); pin++ ) {  
        pin->updateLocation( newLocation );  
        netarray.updatePin( *pin );  
    }  
}
```

Data Distribution

- Use distributed objects derived from serial class as well as aggregate class
- Cell and Net objects are not duplicated
- Prepartitioning phase is used to determine initial assignments of each cell and net.



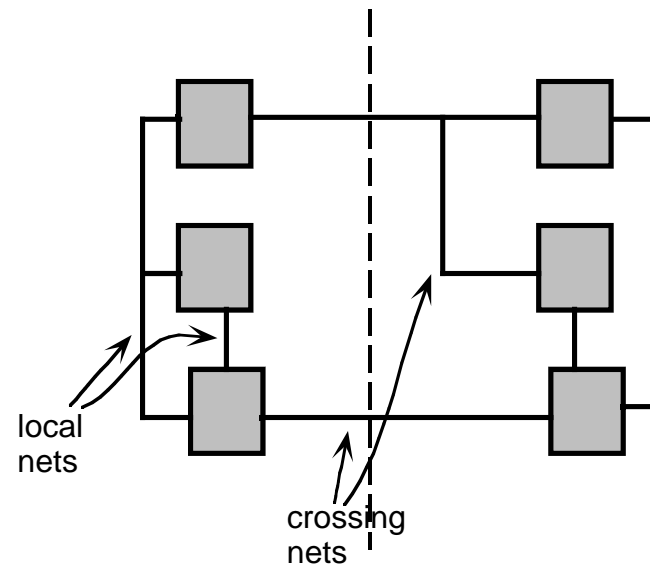
Data Distribution

```
class CellAggr : public ClList, public Aggregate {
    Cell &pickACell();
    // ...
};

Cell &CellAggr::pickACell()
{
    Cell &cell = ClList::pickACell();
    while ( !isLocalCell( cell ) )
        cell = ClList::pickACell();
}
```

Partitioning issues

- Non local data structures
- Certain nets may cross partition boundaries
- These nets are replicated



Updating issues

- When a local copy of a net is modified, we need to update the master copy as soon as possible

```
class NetAggr : public NtList, public Aggregate {
    void updatePin( Pin& );
    class updatePin : public ActorMethod<Pin> {};
    // ...
}

void NetAggr::updatePin( Pin &p )
{
    if ( isLocalNet( p ) {
        NtList::updatePin( p );
    } else {
        updatePin::Continuation cont( ownerOf( p ) );
        cont( p );
    }
}
```

Updating issues

- How do we allow update messages to be processed?

```
class cktAggr : public Circuit, public Aggregate {
    void anneal();
    class anneal : public ActorMethodVoid {};
};
```

```
void CktAggr::anneal()
{
    tryCellMove();
    if ( terminationNotReached() ) {
        CktAggr::anneal::Continuation cont( *this );
        cont();
    } else {
        finishUp();
    }
}
```

Other algorithmic issues

- **Dynamic redistribution to minimize communication as well as improve quality**
- **Several sophisticated error control mechanisms**
- **Memory scalable because of partitioning**

Parallel algorithm

```
if actor on thread 0
  then read circuit and distribute cells and nets to different threads
for each thread
  while termination not reached
    while TimberWolfSC iteration not complete
      attempt move
      attemptsCount++
      if attemptsCount% $U_p$  == 0
        then update pins
      if attemptsCount% $U_r$  == 0
        then update rows
          fix row desires
    adjust  $U_p$  and  $U_r$ 
    update fixed cells
    remove overlaps
    if time to repartition
      then repartition
```

Results

- Sun SparcServer 1000E (shared memory)

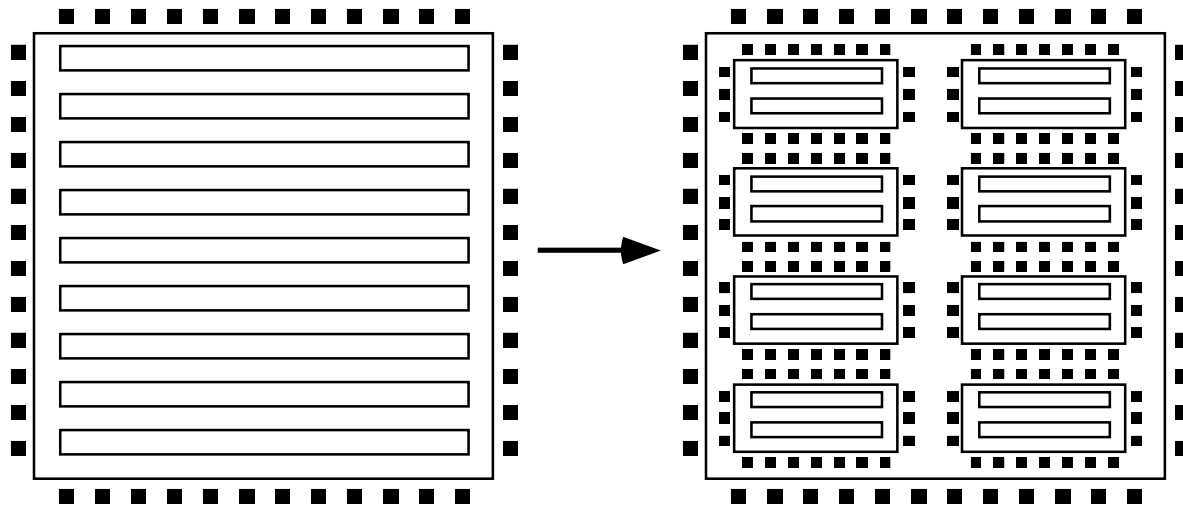
Circuit	1 proc		4 Proc		8 Proc	
	Wirelength	Speedup	Wirelength	Speedup	Wirelength	Speedup
industry1	1.01	1.23	1.06	3.20	1.08	4.36
primary2	1.01	1.11	1.02	3.20	1.06	6.04
biomed	0.92	1.16	1.08	2.20	1.10	5.04
avq.large	1.00	1.22	1.05	2.43	1.09	4.77

Circuit Partitioned Results (cont.)

- Intel Paragon (distributed memory)

Circuit	1 proc		8 Proc		16 Proc	
	Wirelength	Speedup	Wirelength	Speedup	Wirelength	Speedup
industry1	0.95	0.90	1.03	3.85	1.13	6.37
primary2	1.00	0.94	1.04	5.08	1.13	7.13
biomed	–	–	1.03	2.73	1.06	6.62
avq.large	–	–	–	–	1.05	6.23

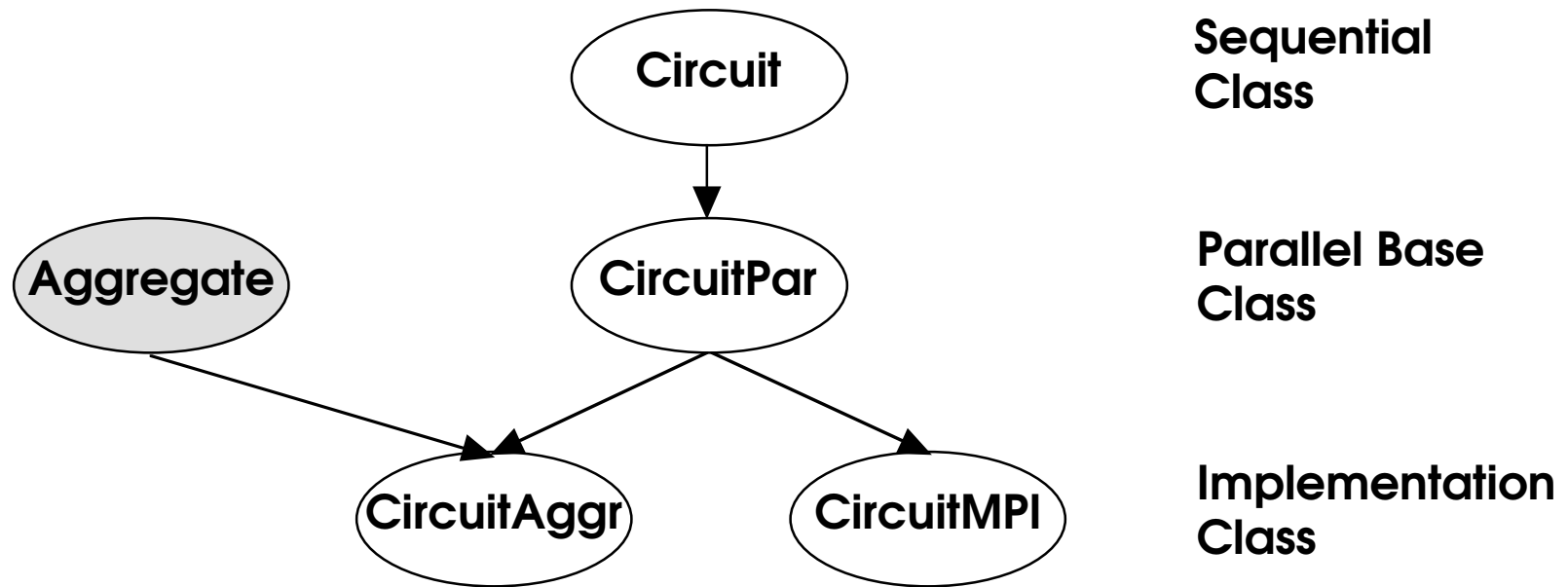
Partitioned Placement



- **SUN SparcServer 1000E**

avq.large	Wirelength	Run Time (s)
TimberWolfSC	11,545,887	79941
TimberWolfSC 8 partitions	16,642,589	9699
mpiPLACE 8 processors	12,406,356	14975

Class Hierarchy



Class Hierarchy

```
class CellPar : public ClList {
    ...
    virtual void sendUpdatePins( UpdateMsg&, int );
};

void CellMPI::sendUpdatePins( UpdateMsg &umsg, int p )
{
    MPI_Send(umsg, sizeof(UpdateMsg), UPDATE_TAG,
             MPI_COMM_WORLD, p);
}

void CellAggr::sendUpdatePins( UpdateMsg &umsg, int p )
{
    NetAggr::UpdatePins::Continuation
        ( nameOfRepresentative(p) ) ( umsg );
}
```

Related work

- **Charm++** - similar, does not support static message typing and is difficult to construct composable libraries
- **pC++** - more suitable to data parallel HPF type models
- **CC++** - task parallelism approach is similar to actor model, but ProperCAD has meta-programmability features that allow the programmer to change the behavior of the run time system.

Conclusions

- **Presented a library for building large parallel programs using standard C++**
- **Allows rapid development of applications with minimal change to base serial code**
- **Example presented for parallel cell placement**
- **Same data structures being effectively reused in timing driven placement and global routing**