AUTOMATIC LOOP PARALLELIZATION

2013-9-27

Presenter: Ali Tarhini

Thesis

Contents

1. Introduction

- With the advance of multi-core and many core systems, parallel programming has become possible for all developers.
- Transforming sequential code to parallel code yields to improvement in overall throughput.
- Manual code transformation is a difficult and error prone task.

Automatic Parallelization

1. Introduction – Problem Area

- Loops are the most important target for parallelization.
  - Loops dominate the execution time.
  - Most programs spend a lot of their running time iterating through one or more compute intensive loops.
  - Typically: 90/10 rule, 10% code is a loop.

Automatic Loop Parallelization

1. Introduction - Motivation

- Available loop parallelization techniques can't utilize parallelism available in code effectively.
  - Long running time
  - Huge processors communication
  - Large memory usage

1. Introduction - Expectations

- A new approach is proposed.
  - Extracts parallelism among different loop iterations and also inside statements of the same iteration.
  - Incurs less processor, memory and time overheads.

2. Introduction - Expectations

Time	Processor 1	Processor 2	Processor 3	Processor 4
T1	S11			
T2	S21			
T3		S12		
T4			S22	
T5		S13		
T6		S23		
T7			S14	
T8		S24		
T9	S15			
T10				S25
T11		S16		
T12			S26	
T13	S17			
T14				S27

Serial Schedule

Time	Processor 1	Processor 2	Processor 3	Processor 4
T1	S11	S21	S22	S13
T2	S25	S17	S27	
T3	S12	S23	S14	S24
T4	S15	S16		
T5	S26			

Parallel Schedule

1. Introduction - Specific Problem

Input

```
Do I=1,7
  S1: A(B(I))=…
  S2: …=A(C(I))
END DO
```

Access Array

I	1	2	3	4	5	6	7
B(I)	3	1	2	4	2	1	6
C(I)	1	4	2	3	8	2	7

Output

Execution Order of Itemsets

Parallel Schedule

2. Background - Dependencies

Data Dependency

- It occurs when the input of a statement requires the output of another previous statement.

Anti Dependency

- It occurs when a statement updates a value that is read by another previous statement.

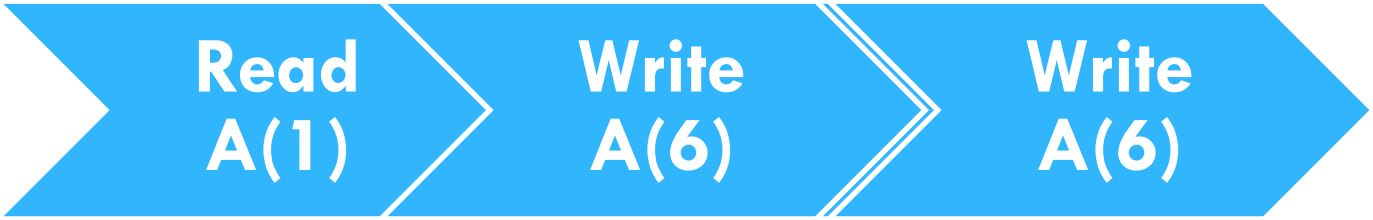
Output Dependency

- It occurs when two statements update the value of the same variable in the program code so that ordering becomes important.

2. Background - Dependencies

Access array showing accesses of array elements across loop iterations.

I	1	2	3	4	5	6	7
Write	3	1	2	4	2	6	6
Read	1	4	2	3	8	2	7



Output Dependency

2. Background - Dependencies

- How to obtain a full parallel schedule of the loop?
  - Using dependency information, build a list of sets containing independent items.
  - Independent items in each set can be executed in parallel.

3. Related Work

- Many algorithms have been developed for parallelizing loops
  - Zhu and Yew
  - Midkiff and Padua
  - Leung and Zahorjan
  - Chen, Yew and Torellas
  - Xu and Chaudhary
  - Kao, Yang and Tseng

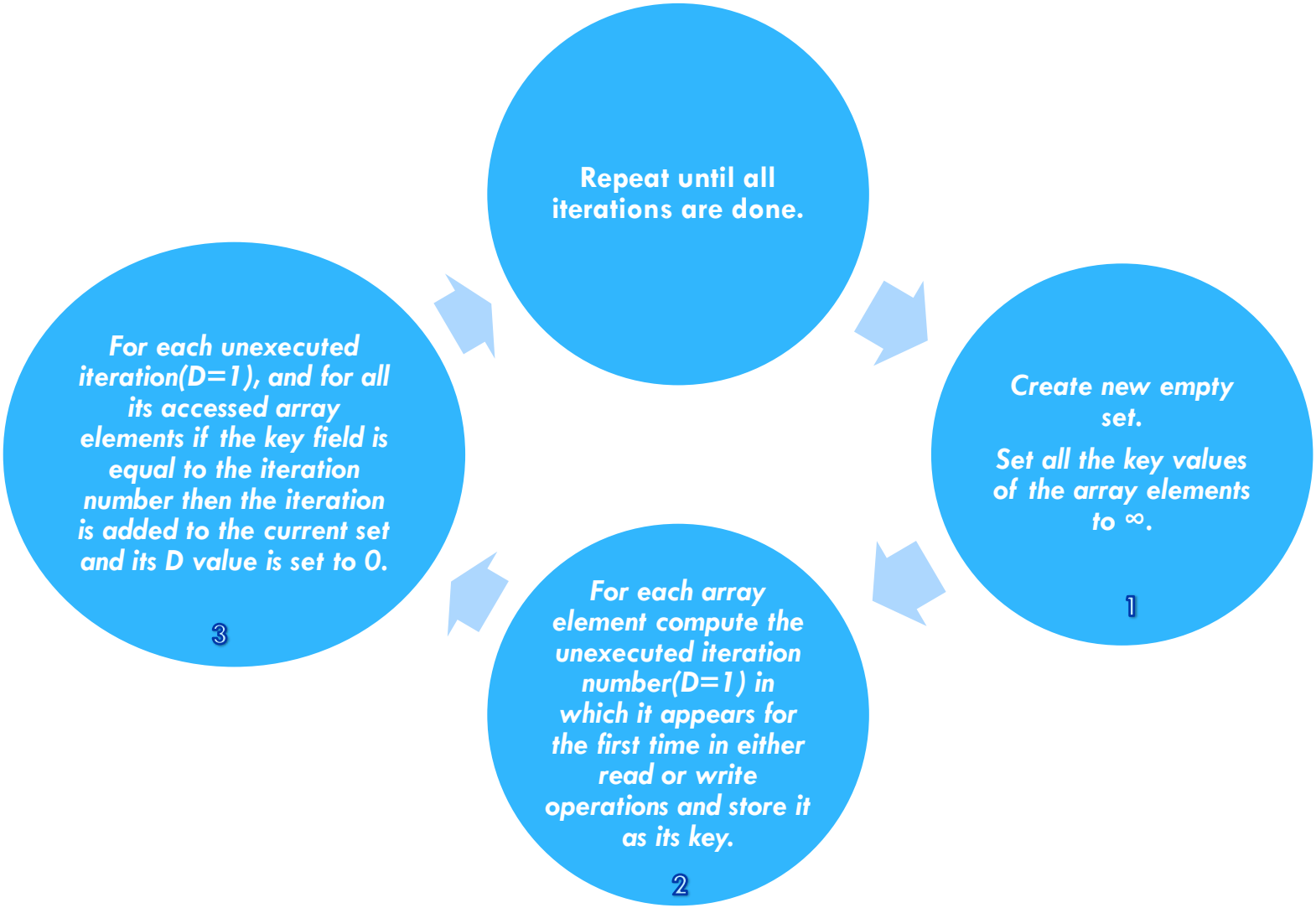
Zhu and Yew

- The earliest approach to parallelize loops was proposed by Zhu and Yew.
- The algorithm consists of two phases which are:
  - The registration phase where a parallel schedule is constructed.
  - The execution phase where the obtained schedule is executed.

Zhu and Yew

- The algorithm stores
  - For every array element a key value.
  - For every loop iteration a D value to indicate if the iteration is done.

Zhu and Yew Algorithm



Zhu and Yew Example

I	1	2	3	4	5	6	7
Write	3	1	2	4	2	1	6
Read	1	4	2	3	8	2	7

1. • 1, 3, 7
2. • 2, 5
3. • 4, 6

Round1

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	∞	∞	∞	∞	∞	∞	∞	∞

I	1	2	3	4	5	6	7
D	1	1	1	1	1	1	1

→

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	1	3	1	2	X	7	7	5

I	1	2	3	4	5	6	7
D	0	1	0	1	1	1	0

Round2

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	∞	∞	∞	∞	∞	∞	∞	∞

I	1	2	3	4	5	6	7
D	0	1	0	1	1	1	0

→

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	2	5	4	2	X	X	X	5

I	1	2	3	4	5	6	7
D	0	0	0	1	0	1	0

Round3

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	∞	∞	∞	∞	∞	∞	∞	∞

I	1	2	3	4	5	6	7
D	0	0	0	1	0	1	0

→

A(I)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)
Key	6	6	4	4	X	X	X	X

I	1	2	3	4	5	6	7
D	0	0	0	0	0	0	0

Zhu and Yew

Advantages	Disadvantages
Simple	Many memory accesses and rounds
	No Intra-Parallelization
	No Concurrent Reads

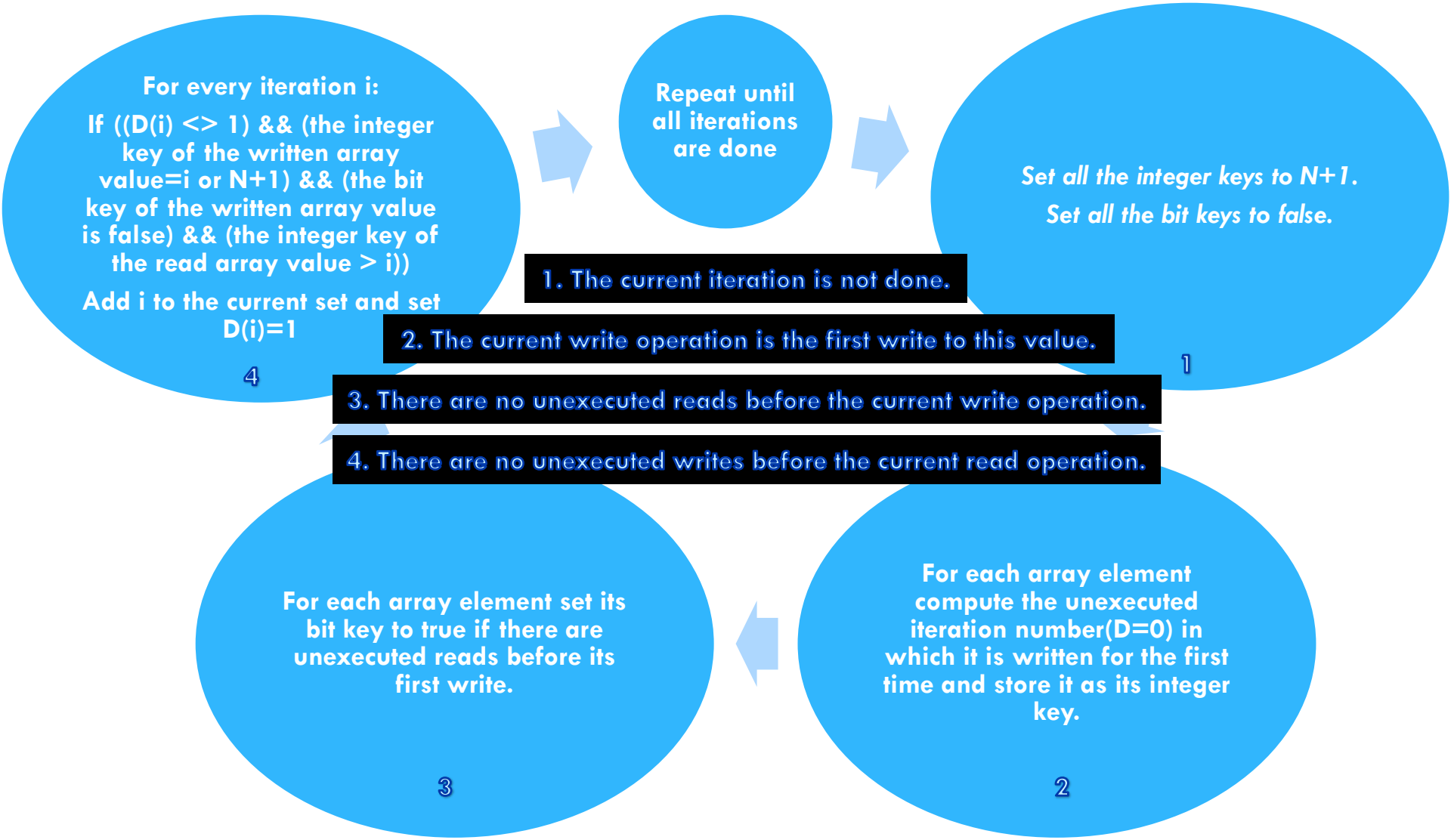
Midkiff and Padua

- It uses a similar scheme to Zhu and Yew but with one improvement by allowing concurrent reads across different iterations.
- The algorithm consists of two phases which are:
  - The registration phase where a parallel schedule is constructed.
  - The execution phase where the obtained schedule is executed.

Midkiff and Padua

- The algorithm stores for each array element 2 keys:
  - The integer key value contains the number of the iteration that does the first write operation to the array element.
  - The bit key value is set to true if there are any iterations that have read operations on the array element preceding the iteration that performs the first write operation on the array element.
- The algorithm also uses an array D which has the same size as the loop size.

Midkiff and Padua Algorithm



Midkiff and Padua Example

I	1	2	3	4	5	6	7	8	9
Write	3	6	3	4	4	3	5	5	5
Read	1	3	1	1	2	2	2	1	1

1. 1, 4, 7
2. 2, 5, 8
3. 3, 9
4. 6

Round1

I	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
V	10	10	1	4	7	2
B	F	F	F	F	F	F

I	1	2	3	4	5	6	7	8	9
D	1	0	0	1	0	0	1	0	0

Round2

I	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
V	10	10	3	5	8	2
B	F	F	T	F	F	F

I	1	2	3	4	5	6	7	8	9
D	1	1	0	1	1	0	1	1	0

Round3

I	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
V	10	10	3	10	9	10
B	F	F	F	F	F	F

I	1	2	3	4	5	6	7	8	9
D	1	1	1	1	1	0	1	1	1

Round4

I	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
V	10	10	6	10	10	10
B	F	F	F	F	F	F

I	1	2	3	4	5	6	7	8	9
D	1	1	1	1	1	1	1	1	1

Midkiff and Padua

Advantages	Disadvantages
Concurrent Reads	Many memory accesses and rounds
	No Intra-Parallelization

Leung and Zahorjan

- The algorithm performs the inspector phase in parallel.
  - The iterations are divided among the available processors and each processor constructs its own sub schedule for its assigned iterations.
  - Finally the sub schedules of all the processors are concatenated together to form the synchronization schedule.

Leung and Zahorjan Example

- Assume we have 15 iterations in a loop to be divided among 3 processors.

Section	1	2	3
Iterations	1, 2, 3, 4, 5	6, 7, 8, 9, 10	11, 12, 13, 14, 15

Section	1		2		3		
Number in sub-schedule	1	2	1	2	1	2	3
Number in overall schedule	1	2	3	4	5	6	7
Iterations	1, 2, 5	3, 4	6, 7, 9	8, 10	11, 14	12	13, 15

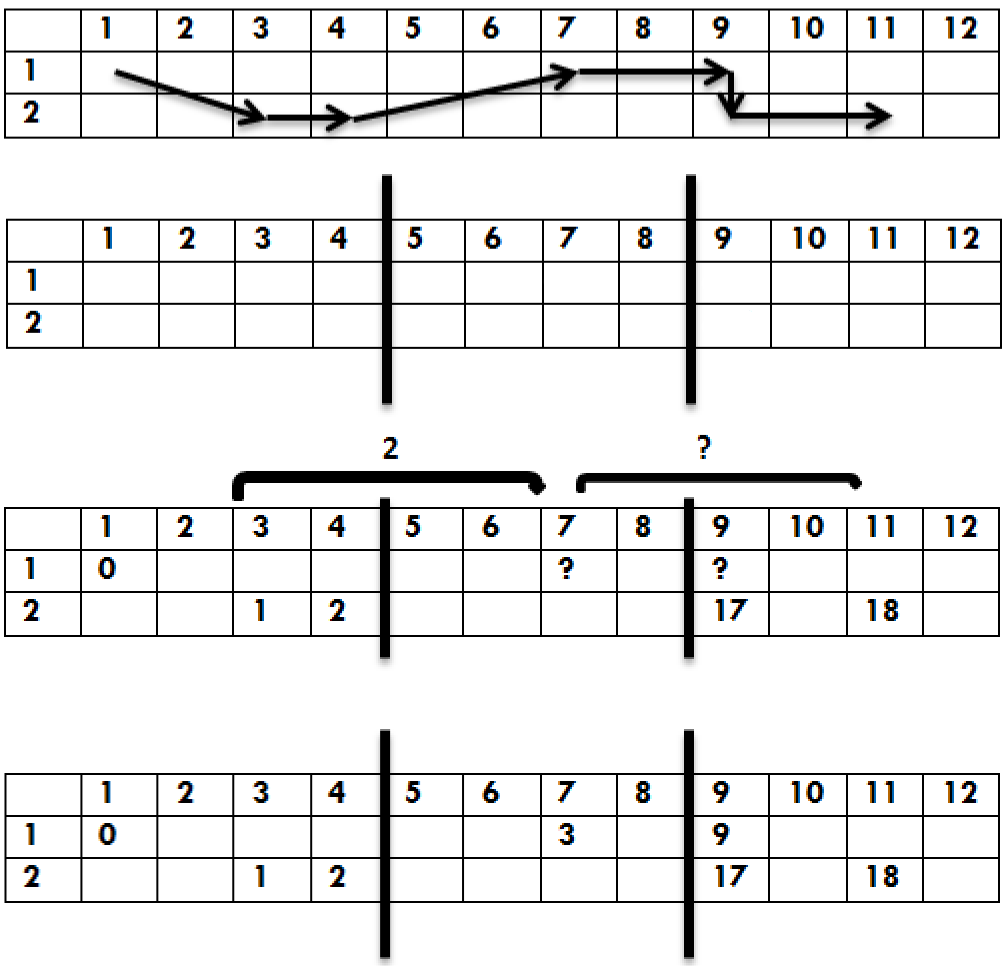
Leung and Zahorjan

Advantages	Disadvantages
Faster Inspection Phase	**It decreases the degree of parallelization.**

Chen, Yew and Torellas

- In this algorithm a ticket table for every dependence chain is constructed.
  - The ticket table stores the sequence numbers for accessing the array elements in the correct order.
- This algorithm consists of 2 phases:
  - the inspector phase
  - and the executer phase

Chen, Yew and Torellas Inspector Phase



[illegible] head of [illegible] dependence chain which means that it is unresolved.

Chen, Yew and Torellas Inspector Phase

- For each iteration the statement gets added to the set if its array element for this iteration has a key equal to the ticket value.
- If the array element has a key value less than the ticket value of the first element in the set assigned to this processor, then the key is set to this value, otherwise it is incremented by 1.

Chen, Yew and Torellas

Advantages	Disadvantages
Intra-Parallelization	Huge communication overhead among the processors.
It performs the inspection phase in parallel.	No Concurrent Reads

Xu and Chaudhary

- A timestamp algorithm for parallelizing loops which allows concurrent reads.
- The algorithm uses a 2 dimensional array A[i][j], where
  - i denotes the sequence number.
  - j denotes the read group size if the operation is a read operation otherwise it is set to zero.

Xu and Chaudhary

- A(i,j) denotes the array element of the current iteration and A(m,n) refers to its direct predecessor.
- L-closure and R-closure are used to indicate the beginning and the end of a read group, so that they can be executed concurrently.
- The u value denotes an undefined value.
- r denotes the processor number of the current iteration.
- g(r) represents the total number of elements contained in all the previous processors.
- h(r) denotes the number of references in the same block r.

Xu and Chaudhary Step 1

Head

Xu and Chaudhary Example (1)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3	13	10	15	0	8	10	10	1	10	10	15	3	15	11	0
W	15	5	5	14	10	14	12	11	3	12	4	8	3	10	10	3

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3		10				10	10		10	10		3			
W					10				3				3	10	10	3

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 (1,u)		10 (1,u)				10 (L,-15)	10 (-13,u)		10 (u,-21)	10 (-19,u)		3 (u,R)			
W					10 (u)				3 (u)				3 (26)	10 (u)	10 (26)	3 (27)

Xu and Chaudhary Step 2

Write

- Timestamp(i,j,0)=Timestamp(m,n,0) +1 if its predecessor performs a write.
- Timestamp(i,j,0)=g(r') + 2 otherwise

Read

- Timestamp(i,j,1)=Read Group Size
- Timestamp(i,j,0)=g(r')+h(r')+1 if stamp(m,n,0)=u
- Timestamp(i,j,0)=Timestamp(m,n,0)+1 otherwise

Xu and Chaudhary Example (2)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 (1,u)		10 (1,u)				10 (L,-15)	10 (-13,u)		10 (u,-21)	10 (-19,u)		3 (u,R)			
W					10 (u)				3 (u)				3 (26)	10 (u)	10 (26)	3 (27)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 (1,1)		10 (1,1)				10 (10,4)	10 (10,4)		10 (10,4)	10 (10,4)		3 (18,1)			
W					10 (2)				3 (2)				3 (26)	10 (18)	10 (26)	3 (27)

Xu and Chaudhary Step 3

Write	Time(k)=Time(k)+1
Read	Time(k)=Time(k) + 1/Timestamp(i,j,1)

Xu and Chaudhary Example (3)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 (1,1)		10 (1,1)				10 (10,4)	10 (10,4)		10 (10,4)	10 (10,4)		3 (18,1)			
W					10 (2)				3 (2)				3 (26)	10 (18)	10 (26)	3 (27)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 1		10 1				10 2.25	10 2.5		10 2.75	10 3		3 3			
W					10 2				3 2				3 4	10 4	10 5	3 5

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3 1		10 1				10 3	10 3		10 3	10 3		3 3			
W					10 2				3 2				3 4	10 4	10 5	3 5

Xu and Chaudhary

Advantages	Disadvantages
Concurrent Reads	Huge communication overhead among the processors.
It performs the inspection phase in parallel.	Time Overhead
Intra-Parallelization	Memory Overhead

Kao, Yang and Tseng

- A technique for parallelizing loops by building a table containing the iteration number where a given element is read and written for each iteration.
- For every loop iteration:
  - The array element A(B(i)) is written at this iteration:
    - W(A(B(i)))=i
  - The array element A(C(i)) is read at this iteration:
    - R(A(C(i)))=i
  - WF(i)= WF(Max[W(A(B(i-1))), R(A(B(i-1))), and W(A(C(i-1)))]) + 1.

Kao, Yang and Tseng Example

I	1	2	3	4	5	6	7	8
W	3	7	3	5	5	8	7	7
R	2	3	2	2	5	3	2	2

1		1	2	3	4	5	6	7	8	WF(I)
1	W									
	R									
2	W									
	R									
3	W									
	R									
4	W									
	R									
5	W									
	R									
6	W									
	R									
7	W									
	R									
8	W									
	R									

Sets	Iterations
1	1, 4
2	2, 5
3	3, 7
4	6, 8

Kao, Yang and Tseng

Advantages	Disadvantages
Concurrent Reads	**Memory Overhead**
	No Intra-Parallelization

4. Proposed Approach

- Our proposed approach for parallelizing loops is based on extracting independent sets.
- The uniqueness of our algorithm lies in optimizing the memory storage of dependencies and it does not incur processor overhead.
- It consists of two phases:
  - The inspector phase where a parallel schedule is constructed.
  - The executer phase where the parallel schedule is executed.

4. Proposed Approach

- The key idea behind our algorithm is that if statement S1 depends on another statement S2, then S1 is assigned a larger sequence number to ensure that S1 is executed after S2.
- S2 is directly identified from a dependency table that stores the last operation that used an array element.

4. Proposed Approach

Input

```
Do I=1,7
  S1: A(B(I))=…
  S2: …=A(C(I))
END DO
```

Access Array

I	1	2	3	4	5	6	7	…	N
Write B(I)	3	1	2	4	2	1	6	…	…
Read C(I)	1	4	2	3	8	2	7	…	…

Output

Parallel Schedule

4. Proposed Approach-Inspection

- The inspection phase consists of 2 steps:
  - Sequence Numbering: every statement is assigned the sequence number of the statement it depends on incremented by 1, except in the case when there are consecutive reads then both are assigned the same sequence number.
  - Dependency Storage: the index of the current statement and the type of the operation are stored in a dependency table for the used array element in order to keep track of dependencies.

4. Step 1 Sequence Numbering

- The sequence array is used for ordering the execution of the loop statements.
- Assuming we have a loop of size N and the loop body contains 2 statements (R & W), we create a sequence array having a size of 2N.

	0	1	2	3	4	5	6	7	8	…	2N-1
S											

4. Step 1 Sequence Numbering

- If a statement S1 is dependent on another statement S2 then S1 is not allowed to execute before S2 terminates.
- S1 is assigned a sequence number greater than S2 by 1.
- For example, if statement of index 9 is dependent on statement of index 2 then:
  - S(9)=S(2) + 1

4. Step 2 Storing Dependencies

- The dependency array is used to identify the dependent statements of a given loop.
- Assuming we have an input array of size M, we create a new array for storing dependencies having a size of 2M.
  - I stores the index of the last statement that used the array element.
  - T is used to store the type of the last operation that used the array element if it is a *Read* or *Write*.

Array Elements

	1	2	3	4	5	6	7	8	…	M
I										
T										

4. Proposed Approach-Inspection


```
For k=0 to StatementsCount
        IF D(AT(k)).I=-1Then
                S(k)=1
        Else
                IF Operation is read Then
                        IF D(AT(k)).T is read Then
                                S(k)=S(D(AT(k)).I)
                        Else
                                S(k)=S(D(AT(k)).I)+1
                Else
                        S(k)=S(D(AT(k)).I)+1
                ENDIF
        ENDIF
        D(AT(k)).I=k
        D(AT(k)).T=Operation
NEXT
```

Access Array

Sequence Table

Dependency Table

4. Example

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	3	13	10	15	0	8	10	10	1	10	10	15	3	15	11	0
W	15	5	5	14	10	14	12	11	3	12	4	8	3	10	10	3

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
W	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R			10				10	10		10	10					
W					10									10	10	

4. Example

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	2	10 4	6	8	10	10 12	10 14		10 18	10 20					
W	1	3	5	7	10 9	11								10 27	10 29	

Index

4

S(4)=1
- D(10).I=4
- D(10).T='R'

9

S(9)=S(4)+1=2
- D(10).I=9
- D(10).T='W'

12

S(12)=S(9)+1=3
- D(10).I=12
- D(10).T='R'

14

S(14)=S(12)=3
- D(10).I=14
- D(10).T='R'

18

S(18)=S(14)=3
- D(10).I=18
- D(10).T='R'

20

S(20)=S(18)=3
- D(10).I=20
- D(10).T='R'

27

S(27)=S(20)+1=4
- D(10).I=27
- D(10).T='W'

29

S(29)=S(27)+1=5
- D(10).I=29
- D(10).T='W'

4. Example

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	1	1	1	1	1	2	2	1	1	2	1	2	3	1	3	1
Index	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S	1	2	3	2	3	1	2	2	3	4	2	4	2	5	1	5

Execution Order of Itemsets	Cell Indices
1	0, 1, 2, 3, 4, 7, 8, 10, 13, 15,16,21,30
2	5, 6, 9, 11, 17,19,22,23,26,28
3	12,14,18,20,24
4	25,27
5	29,31

4. Proposed Approach-Execution Phase

- In the execution phase, items of the same set are run concurrently, and when one itemset finishes its execution, it allows the next itemset to execute.
- This is done until all of the sets get executed.

5. Experiments

- We implemented all of the algorithms discussed in the related work section using vb.net.
- We used 3 loops found in the *Perfect Club Benchmark* for testing.
- The experiments were intended to show the difference in performance between proposed approach and the state of the art algorithms in terms of
  - Time taken to build the parallel schedule
  - CPU Consumption
  - Memory Usage
  - Thread Utilization

5. Experiments-Perfect Club Benchmark

- An acronym for PERFormance Evaluation by Cost-effective Transformations.
- Perfect was created at UIUC's Center for Supercomputing Research and Development (CSRD) in an effort to focus supercomputer performance evaluation at the applications level.

5. Experiments(Inputs)

The three Loops were used by the other algorithms for testing.

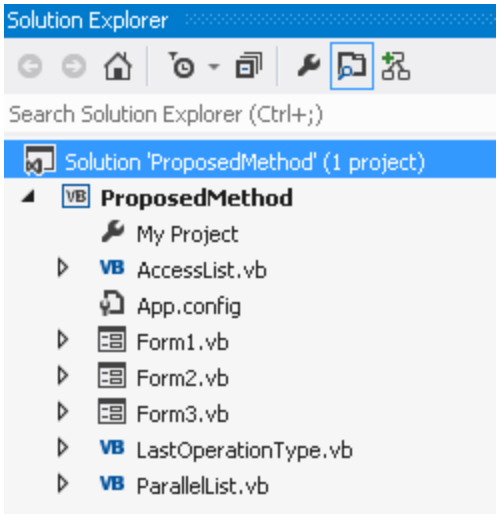
	Code	Loop Size	Array Size	Inputs
Loop1	DO I=1, N DO J=1, N A(S(I))=A(T(I)) ENDDO ENDDO	**250x250**	**1000**	**S(I)=2I** **T(I)=2I+2J+1**
Loop2	DO I=1, N A(S(I))=A(T(I)) + 100 ENDDO	**8000**	**24000**	**S(I)=3I** **T(I)=I+2**
Loop3	DO I=1, N A(S(I))=A(T(I)) + 100 ENDDO	**8000**	**24000**	**S(I)=I+2** **T(I)=3I**

5. Experiments(Setup)

- Implementations of all the algorithms was executed on the same PC.
- Hardware Specs:

5. Experiments(Setup)

- We implemented our approach using the vb.net language.




```
Public Class AccessList
    Public Enum OpType As Byte
        Write = 0
        Read = 1
    End Enum

    Public T As OpType
    Public List() As Integer
End Class

Public Class LastOperationType
    Public Enum OpType As Byte
        Write = 0
        Read = 1
    End Enum

    Public T As OpType
    Public I As Integer = -1
    Public J As Integer = -1
End Class

Public Class ParallelList
    Public iList As New List(Of Integer)
End Class
```

5. Experiments(Setup)


```
Dim AL As New List(Of AccessList)
Dim D As New List(Of LastOperationType)
Dim N As Integer = 15
Dim AN As Integer = 16
Dim S(2 * N - 1) As Integer
Dim ar(AN - 1) As Integer
Dim ResultSet(0) As ParallelList

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load
    InitializeAccessList()
    InitializeLastOperationType()
    BuildSchedule()
    ExecuteLoop()
End Sub


Public Sub InitializeLastOperationType()
    For i As Integer = 0 To AN
        Dim l As New LastOperationType
        D.Add(l)
    Next
End Sub
```

```
Public Sub InitializeAccessList()
    Dim AL1 As New AccessList
    AL1.T = 0 'w
    Dim l(N) As Integer
    AL1.List = l
    AL1.List(0) = 15
    AL1.List(1) = 5
    AL1.List(2) = 5
    AL1.List(3) = 14
    AL1.List(4) = 10
    AL1.List(5) = 14
    AL1.List(6) = 12
    AL1.List(7) = 11
    AL1.List(8) = 3
    AL1.List(9) = 12
    AL1.List(10) = 4
    AL1.List(11) = 8
    AL1.List(12) = 3
    AL1.List(13) = 10
    AL1.List(14) = 10
    AL1.List(15) = 3
    Dim AL2 As New AccessList
    AL2.T = 1 'r
    Dim l2(N) As Integer
    AL2.List = l2
    AL2.List(0) = 3
    AL2.List(1) = 13
    AL2.List(2) = 10
    AL2.List(3) = 15
    AL2.List(4) = 0
    AL2.List(5) = 8
    AL2.List(6) = 10
    AL2.List(7) = 10
    AL2.List(8) = 1
    AL2.List(9) = 10
    AL2.List(10) = 10
    AL2.List(11) = 15
    AL2.List(12) = 3
    AL2.List(13) = 15
    AL2.List(14) = 11
    AL2.List(15) = 0
    AL.Add(AL2)
    AL.Add(AL1)
End Sub
```

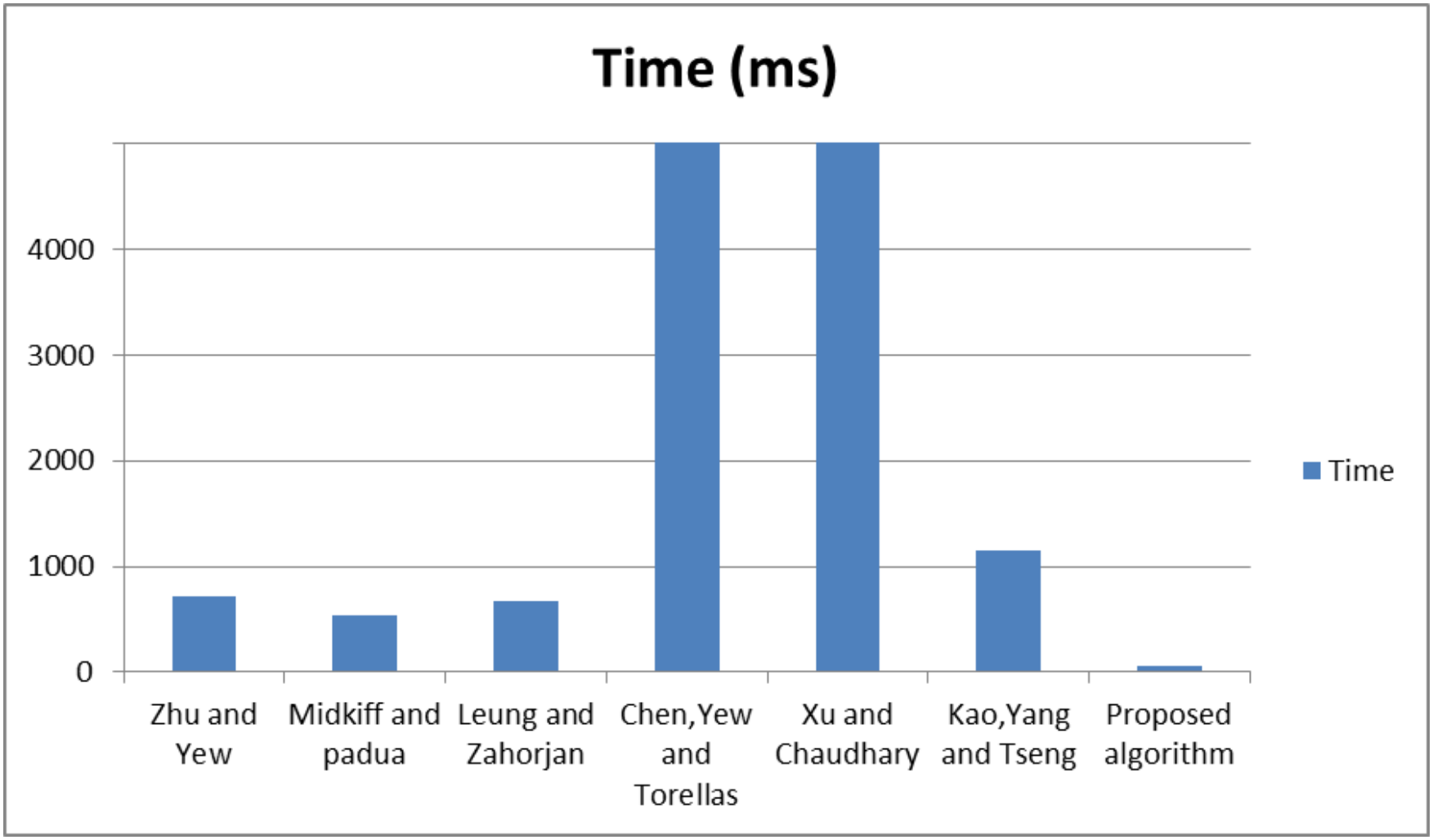
5. Experiments(Setup)


```
Public Sub BuildSchedule()
    ResultSet(0) = New ParallelList
    Dim SequenceNumber As Integer
    Dim index As Integer
    Dim iIndex As Integer
    For i As Integer = 0 To N
        For j As Integer = 0 To AL.Count - 1
            index = i * (AL.Count) + j
            Dim val As Integer = AL(j).List(i)
            iIndex = D.Item(val).I
            If iIndex <> -1 Then
                If AL(j).T = 0 Then
                    SequenceNumber = S(iIndex) + 1
                Else
                    If D.Item(val).T = 1 Then
                        SequenceNumber = S(iIndex)
                    Else
                        SequenceNumber = S(iIndex) + 1
                    End If
                End If
                If ResultSet.Length < SequenceNumber Then
                    ReDim Preserve ResultSet(SequenceNumber - 1)
                End If
                If ResultSet(SequenceNumber - 1) Is Nothing Then
                    ResultSet(SequenceNumber - 1) = New ParallelList
                End If
                ResultSet(SequenceNumber - 1).iList.Add(index)
            Else
                SequenceNumber = 1
                ResultSet(0).iList.Add(index)
            End If
            S(index) = SequenceNumber
            D.Item(val).I = index
            D.Item(val).T = AL(j).T
        Next
    Next
End Sub
```

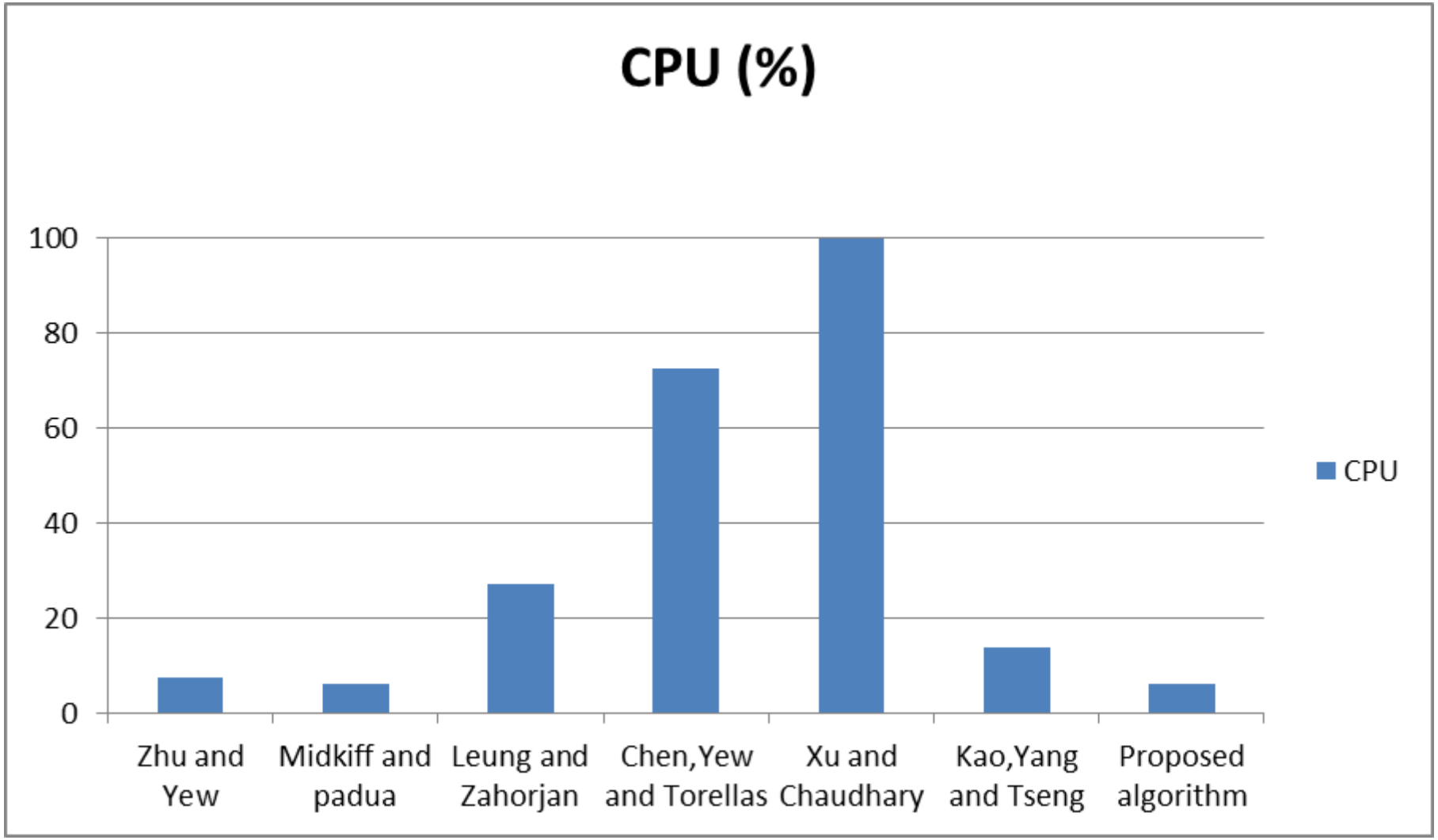
5. Experiments(Results) – Building Schedule

	Time (MS)	CPU (%)	Memory (KB)	Threads
Zhu and Yew	714.33	7.33	115,042.67	13
Midkiff and Padua	540	6	114,385.33	10
Leung and Zahorjan	674.67	27	1,225,864	12
Chen, Yew and Torellas	54,522	72.67	13,636	15
Xu and Chaudhary	92,518	100	36,974.67	15
Kao, Yang and Tseng	1,151.67	13.67	1,301,709.33	10
Our approach	55	6	12,367	8

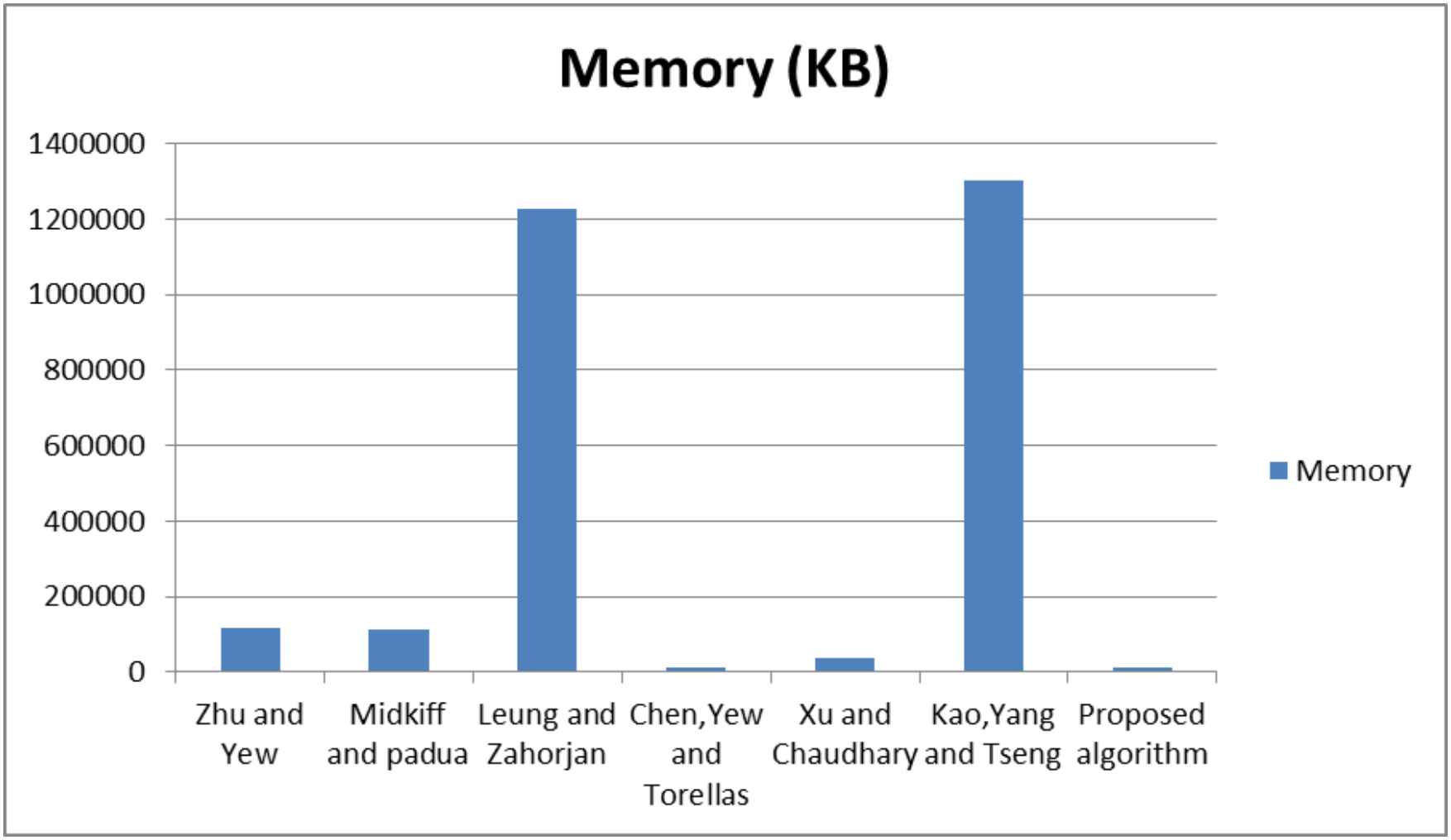
5. Experimental Results – Time Plot



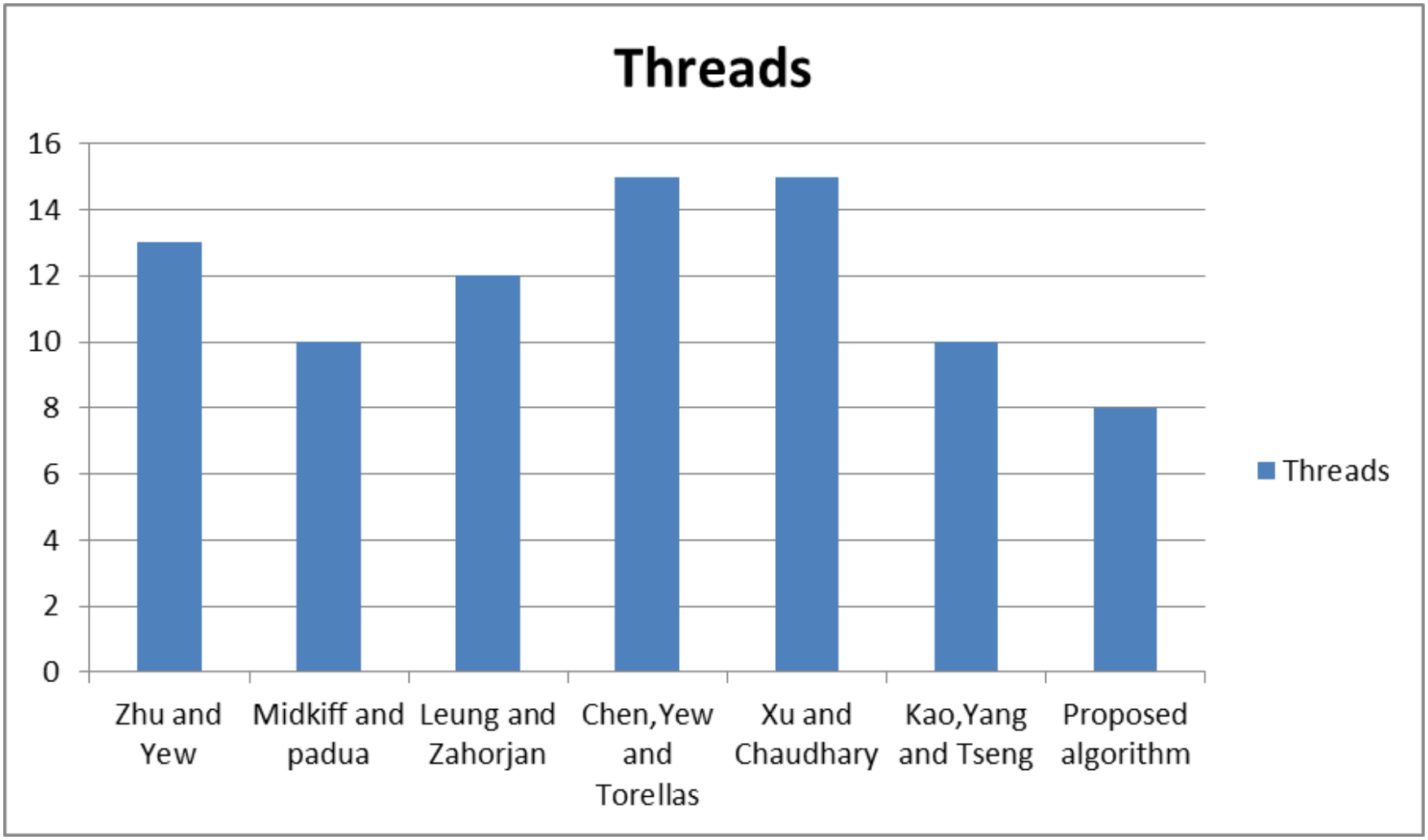
5. Experimental Results – CPU Plot



5. Experimental Results – Memory Plot



5. Experimental Results – Threads Plot



5. Results Interpretation

	Time (MS)	CPU (%)	Memory (KB)	Threads
Proposed Method	**55**	**6**	**12,367**	**8**
Speedup	**Fastest**			
CPU Ratio		**Least cpu overhead**		
Memory Overhead			**Least memory overhead**	
Thread Utilization				**Best utilizing threads**

5. Results (Limitations)

- Our proposed approach and all the algorithms in the related work section use one array in the loop body.
- We need to add more code for the proposed method in order to solve the case when multiple arrays exist that show cross dependencies.

6. Conclusion

- We described a tool that performs automatic loop parallelization.
- We evaluated the effectiveness of our tool by testing it against 6 algorithms and using 3 loops obtained from *perfect club benchmark*.
- In comparing to state of the art loop parallelization techniques, the proposed algorithm achieved a 980% (9.8X) speedup and a 10% reduction in memory usage.

6. Conclusion

- The proposed algorithm has four main advantages:
  - More parallelism by allowing the statements inside an iteration to be executed in parallel.
  - Less time overhead for finding independent sets.
  - Less processor overhead.
  - Less memory overhead.

7. Future Work

- Conduct an empirical evaluation of our vb.net implementation against more algorithms.
- Use arrays that exhibit cross dependencies.
- Use loops from other benchmarks.
- Provide implementation of the proposed approach in other languages.

8. References

- [1] Jiawei Han and Micheline Kmaber, “Data Mining Concepts and Techniques”, Second Edition.
- [2] M.R. Pimple and S.R. Sathe, “Architecture Aware Programming on Multi-Core Systems”, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 6, 2011.
- [3] Sun ZFS Storage Appliances Based on Intel® Xeon® Processors, “Taking Advantage of Multicore/Multiprocessor Technologies to Meet Today’s Storage Needs”, An Oracle White Paper, April 2012.
- [4] Utpal Banerjee, Rudolf Eigenmann, Alexandro Nicolau and David Padua, "Automatic Program Parallelization", IEEE 1993.

8. References

- [5] Michael O'Boyle," Dependence Analysis", School of Informatics, February 2012.
- [6] C. Zhu and P.C. Yew, "A Scheme to Enforce Data Dependence on Large Multi-Processor Systems", IEEE Trans. Software Eng., vol. 13, no. 6, pp. 726-739, June 1987.
- [7] S. Midkiff and D. Padua, "Compiler Algorithms for Synchronization", IEEE Trans. Computers, vol. 36, no. 12, Dec. 1987.
- [8] S.T. Leung and J. Zahorjan, "Improving the Performance of Runtime Parallelization", Proc. Fourth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, pp. 83-91, May 1993.

8. References

- [9] D.-K. Chen, P.-C. Yew, and J. Torrellas, “An Efficient Algorithm for the Run-Time Parallelization of Doacross Loops”, Proc. Supercomputing, pp. 518-527, Nov. 1994.
- [10] Cheng-Zhong Xu and Vipin Chaudhary, “Time Stamp Algorithms for Runtime Parallelization of DOACROSS Loops with Dynamic Dependences”, IEEE Transactions on parallel and distributed systems, vol. 12, no. 5, May 2001.
- [11] Shih-Hung Kao, Chao-Tung Yang, and Shian-Shyong Tseng, “Run-Time Parallelization for Loops”, Proceedings of the 29th Annual Hawaii International Conference on System Sciences, 1996.
- [12] Lawrence Rauchwerger,y Nancy M. Amato, and z David A. Padua, “RunTime Methods for Parallelizing Partially Parallel Loops”.

8. References

- [13] M. Berry and others. The PERFECT club benchmarks: Effective performance evaluation of supercomputers. TR. 827, Ctr. for Supercomputing R.&D., Univ. of Illinois, Urbana, IL, May 1989.
- [14] http://www.parallelsp.com/perfect_club.htm

Thank You