



# **Early Experiences with the Naval Research Laboratory XD1**

**Wendell Anderson**

**Dr. Robert Rosenberg**

**Dr. Marco Lanzagorta**

**Dr. Jeanie Osburn**



# Who we are

## Naval Research Laboratory

- Navy Corporate Lab under the Office of Naval Research

## Center for Computational Science

- Distributed Center under High Performance Computing Modernization Program Office
- Leading edge center evaluating new architectures
- Provide support to NRL and DOD scientists who require High Performance Computing Resources



# Current XD1 Hardware Suite

Twenty-four Chassis System

Each Chassis has six nodes

Each node has

- Two AMD 275 dual core processors
- A Xilinx Vertex II FPGA
- 8 Gigabytes of memory
- 73 Gigabyte SATA drive

15 Gigabyte Fibre channel disk system



## MTA-2 vs XD1 Attributes

	MTA-2	XD1
CLOCK RATE	200 MHZ	2200 MHZ
PROCESSORS	40	576
MEMORY	160 GB	1152 GB
FEATURES	MULTI- THREADING	DUAL CORES FPGAS
MPI ?	NO	YES



# Current XD1 Software Suite

Cray-Modified Suse Linux ( kernel 2.75.5)

GNU and Portland group Fortran/C/C++  
compilers

MPICH 1.2.6 (MPI support)

AMD Core Math Library

Cray Scientific Library

PBSPPro Batch Queuing System



# XD1 Upgrade

Twelve more chassis

- Same processors/memory
- Only 1 chassis has FPGAs (6 Vertex-4s)

Expand available disk space to 30TBs

Additional FPGA software



# XD1 Acceptance Criteria

- 48 hour hardware test
- Run a job across all compute nodes
- Application using FPGA
- Disk I/O bandwidth
- Application Performance
- 24 hour test
- 30-day Acceptance Period



# MTA-2 Application STATIC

Computational Materials tight-binding code

Matrix eigenvalues using LAPACK zhpgv

- 216 1944x1944 matrices
- Most of time in calculation of eigenvalues

MTA parallelize individual matrix calculation

XD1 one matrix per processor

- Little interprocessor communication on XD1

Super XD1 performance (25x)





# MTA-2 Application CAUSAL

Acoustic Propagation with dispersion/attenuation

Finite Difference Time Domain (FDTD) scheme

- Kernels are 1:1 ratio memory accesses to ops
- Memory accesses regular
- Work level differs among grid points

MTA-2 Dynamic assignment a point to a thread

XD1 Static assignment of depths to processor

Significant XD1 performance improvement (3x)



# MTA-2 Application LANCZOS

Computational Materials for quantum systems

Lanczos method to estimate eigenvalues of interest

- Sparse Matrix- Dense Vector Multiply
- Memory accesses irregular

MTA-2 Individual row assigned to thread

XD1 Block of rows assigned to each processor

- Some inter-processor communication (vector)

Small XD1 performance improvement (1.2x)



# MTA-2 Application Results

Application	MTA (secs)	XD1 (secs)	Speedup
STATIC	9529	378	25.2
CAUSAL	936	293	3.2
LANCZOS	336	284	1.2



# AMD 275 Dual Core

<b>Core 1</b>		<b>Core 2</b>	
<b>64 KB I-Cache</b>	<b>64 KB D-Cache</b>	<b>64 KB I-Cache</b>	<b>64 KB D-Cache</b>
<b>1 MB L2 cache</b>		<b>1MB L2 cache</b>	
<b>System Request Interface</b>			
<b>Crossbar</b>			
<b>Integrated DDR Memory Controller</b>			



# Dual Core Applications

## User Applications

ARMS

CAUSAL

LANCZOS

NOZZLE

NRLMOL

STATIC

## TI-05 Applications

AVUS

HYCOM

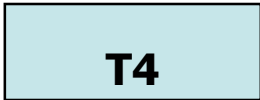
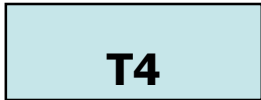
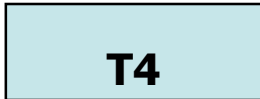
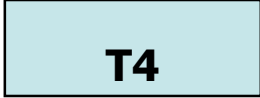
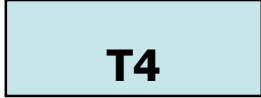
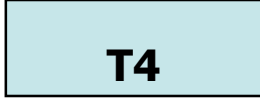
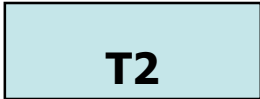
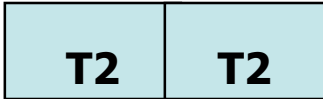
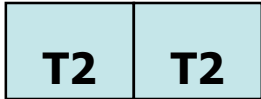
OOCORE

RFCTH2



# Dual Core Efficiency

$$\text{Efficiency} = 100 * (1 - (T4 - T2) / T2)$$

Nodes	$T4 = T2$	$T2 < T4 < 2 * T2$	$T4 = 2 * T2$
<b>N/2</b>			
<b>N/2</b>			
<b>N</b>			



# Dual Core Application Efficiencies (1)

Application	One Core	Two Core	Efficiency %
STATIC	313	450	56
CAUSAL	275	293	93
LANCZOS	771	1371	22
NRLMOL	14283	16260	90
ARMS	2090	2524	79



## Dual Core Application Efficiencies (2)

Application	One Core	Two Core	Efficiency %
NOZZLE	27524	27286	101
AVUS	1197	963	120
HYCOM	823	849	97
OOCORE	5274	7716	54
RFCTH2	279	448	39





# I/O Importance

- Read input data
- Write Restart File
- Write Output Data
- Write/Read Temporary Files



## Disk I/O Rates (MBs)

Number of nodes	Lustre Read	Lustre Write	Local Read	Local Write
1	206	165	40	58
2	325	324	98	105
4	629	646	114	209
8	724	709	273	406
16	892	862	374	804
32	859	893	720	1565



# Scaling Applications

AVUS

HYCOM

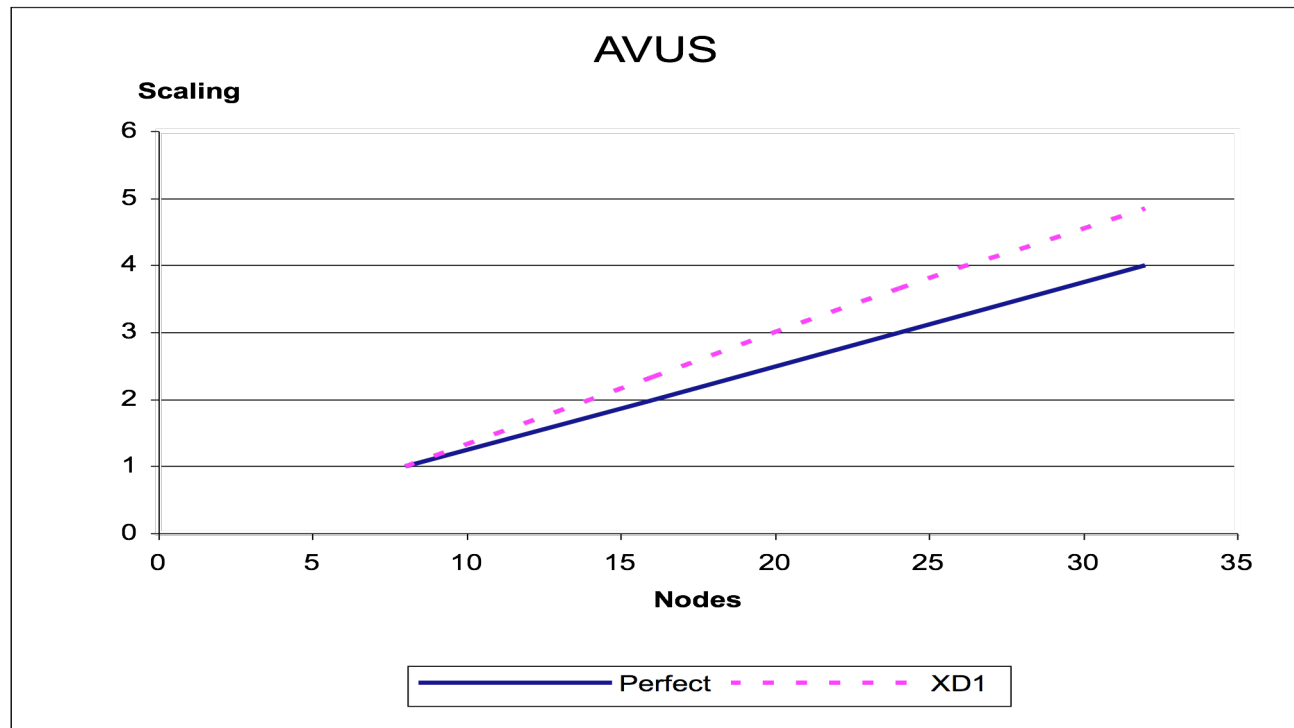
OOCORE

NOZZLE

RFCTH2

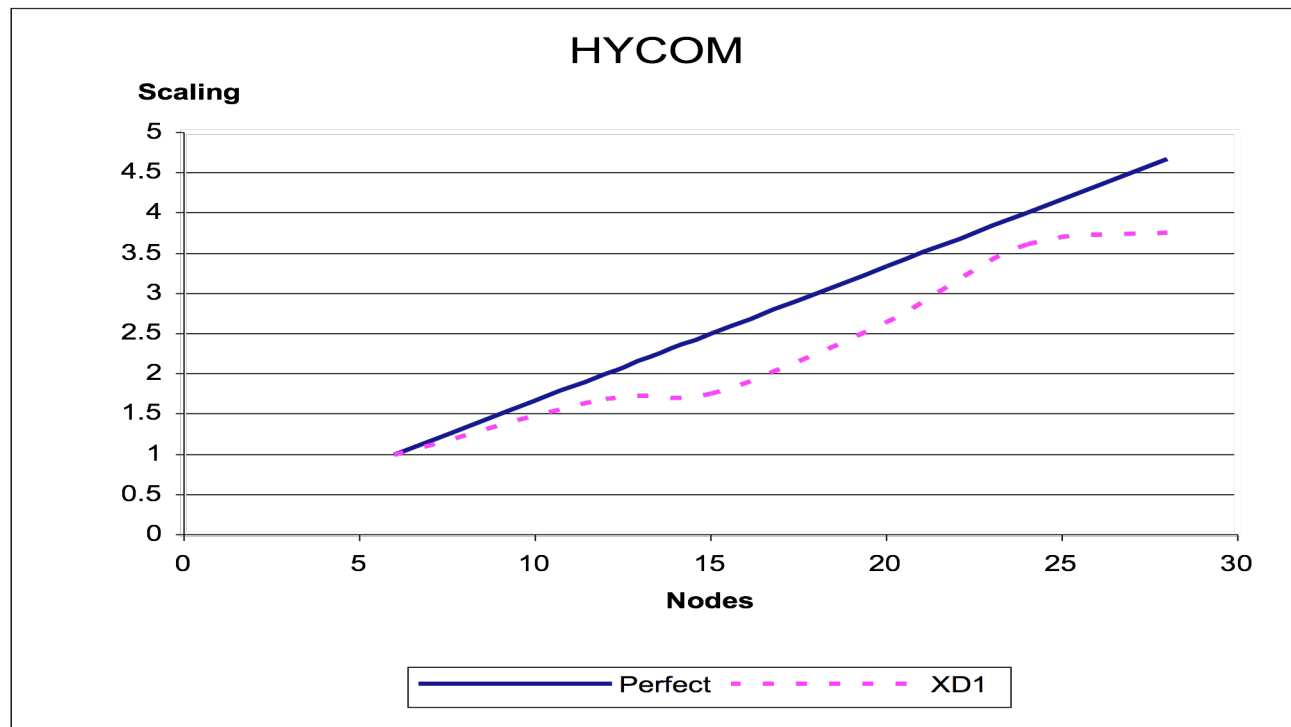


# AVUS Scaling



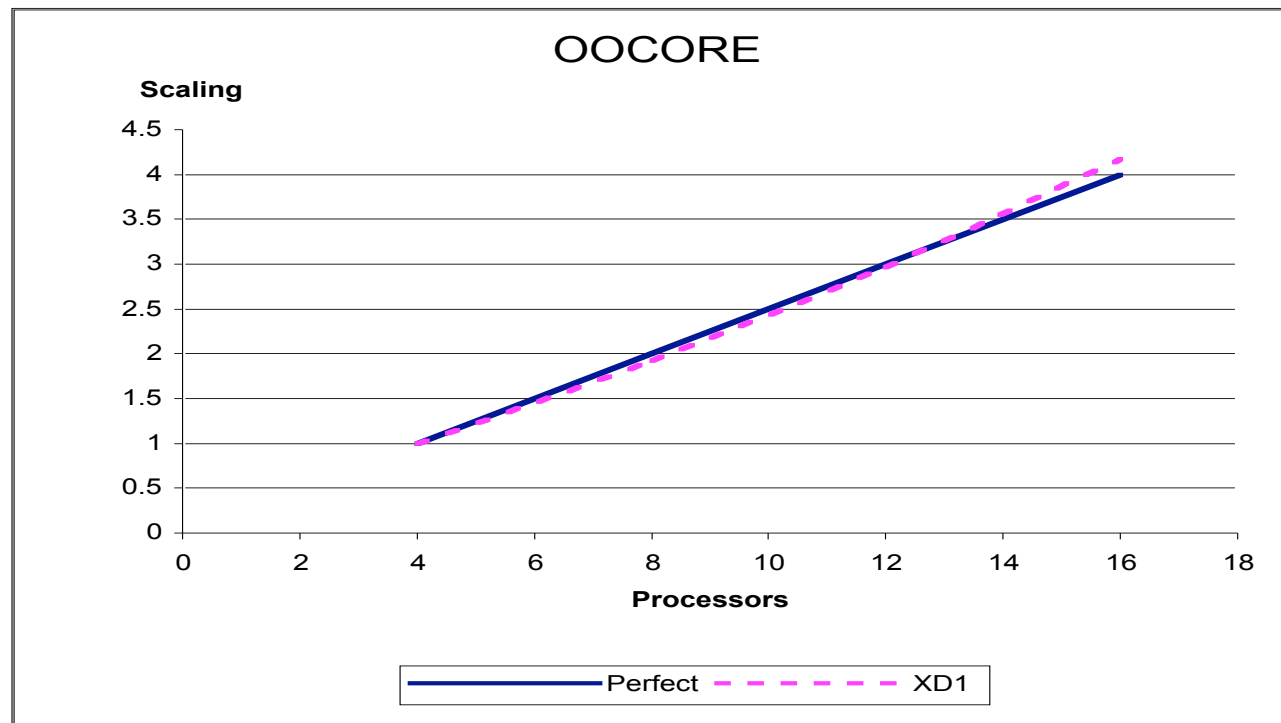


# HYCOM Scaling



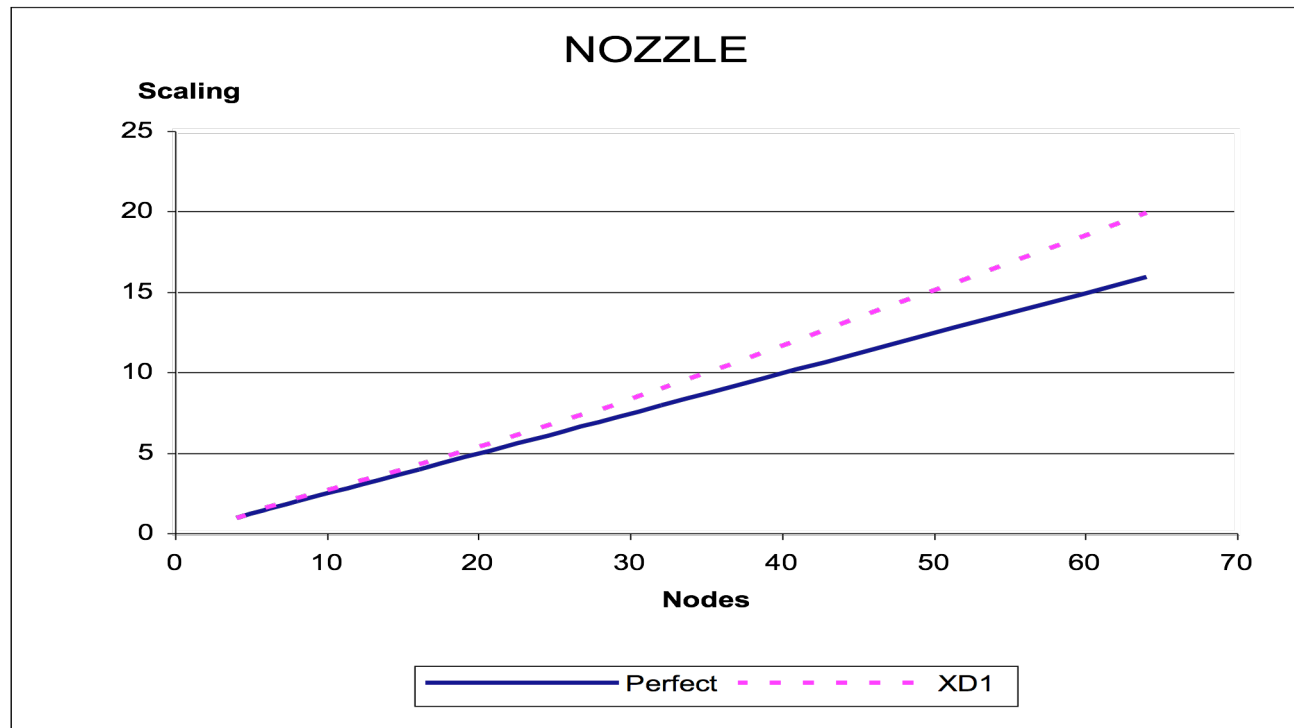


# OOCORE Scaling



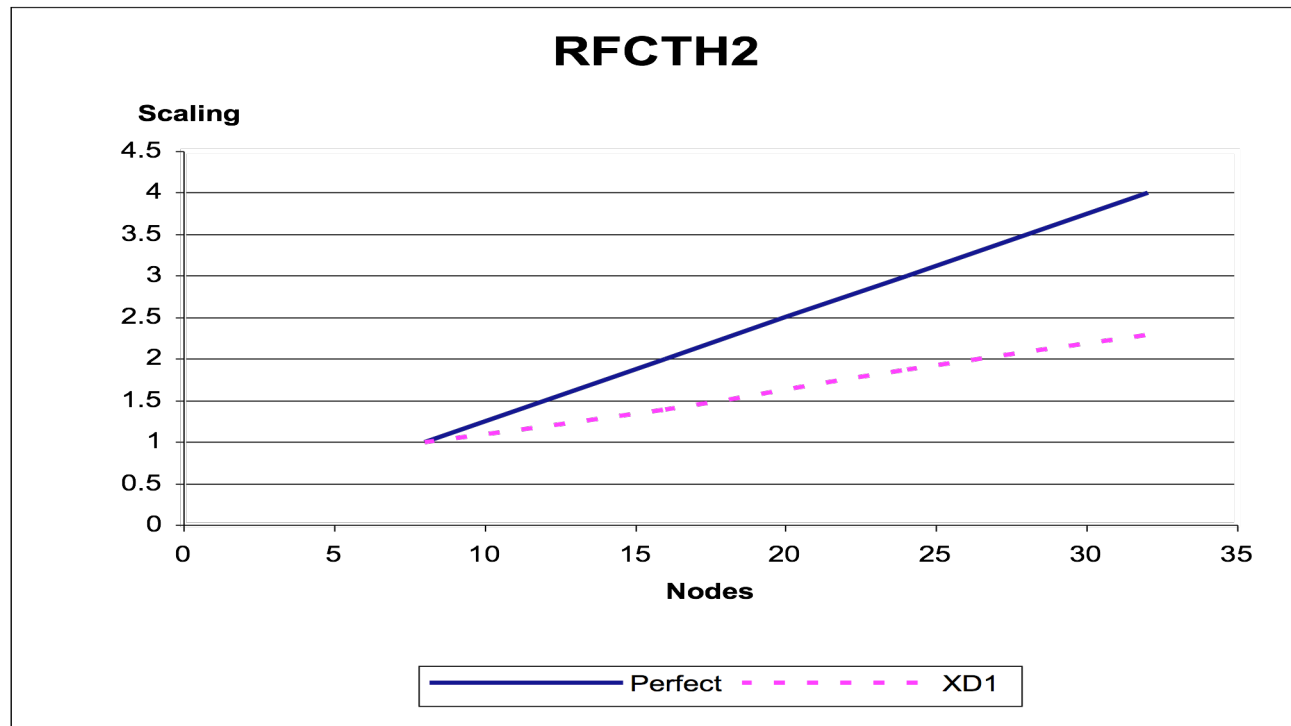


# NOZZLE Scaling





# RFCTH2 Scaling







# HPC FPGA Application Areas

- Astrophysics
- Bioinformatics
- Computational Chemistry
- Computational Fluid Dynamics
- Cryptography
- Electro-magnetics
- Hyper- spectral Imaging
- Network Monitoring
- Signal Processing



# FPGA Cores

- Fast Fourier Transforms
- Gaussian Random Number Generator
- Smith-Waterman
- Uniform Random Number Generator



# Implementing Specific Algorithms

Is the code suited to an FPGA?

- Is there a kernel that takes up most of time?
- Can we use fixed point vs floating point?
- Can we pipeline the code?
- Can we build multiple copies?
- Does the kernel have complicated functions?
- Are there lots of if statements?

Is there another algorithm that is more suited to an FPGA?



# Hyperspectral Image Processing

Determine color mapping for 1 million points of dimension between 64 and 224.

Determine Color Mapping on a set of ten thousand Landmark points

Determine color for each point by interpolating three or four nearest neighbors

Most of time is spent in finding nearest neighbors.



## C Code

$x[i][j]$  is a vector of floats containing the center of a hypersphere  
 $p[l][j]$  is a vector of floats containing the coordinates of a point  
 $r[i]$  is a float the radius of the hypersphere  
flag=FALSE;  
sum=0.0;  
for (j=0; j< NUMDIM; j++) {  
    diff=x[i][j]-p[j];  
    sum=sum+diff\*diff;  
}  
if (sqrt(sum) < r[i]) flag=TRUE ;



# FPGA Friendly C Implementation

Data can be considered 16 bit

Comparison on radius squared removes sqrt

Would a hyper-cube be better?

- Removes multiply per dimension
- Reduces number of bits

Can we do a simple test to see what might happen with performance?



# New C Code

$x[i][j]$  is a vector of floats containing the center of a hypercube  
 $p[l][j]$  is a vector of floats containing the coordinates of a point  
 $r[i]$  is a float containing one half the length of a side of the of the  
hypercube

```
flag=TRUE;
```

```
for (j=0; j< NUMDIM; j++) {
```

```
    testlo=x[i][j]-r[i];
```

```
    testhi=x[i][j]+r[i];
```

```
    if (!((testlo < p[k][j]) && (p[k][j] < testhi))) flag=FALSE;
```

```
    j++;
```

```
}
```



# Opteron vs Mitrion C timings

Dim	Opteron	Mitrion	Speedup
1	85.8	6.6	13
5	283.9	11.4	24
10	591.8	17.7	33
15	889.4	24.7	36
20	1176.7	30.11	39
24	1425.3	35.43	40





# Conclusions

The NRL XD1 has demonstrated significant performance improvements through

- Opteron Dual Cores
- Lustre Disk File System
- High speed interconnect
- Field Programmable Gate Arrays



# Acknowledgement

This work was performed on the Cray XD1 at NRL that was provided under the auspices of the DOD High Performance Computing Modernization Program office.



# Acknowledgements

The authors also thank the following scientists for their willingness to share their expertise and codes

- Dr. Spiro Antiochos
- Dr Chris Bachmann
- Dr Tunna Baruah
- Dr. Richard Devore
- Dr. Stephen Hellberg

Dr. Michael Mehl

Dr. Guy Norton

Dr. Mark Pederson

Dr. Alan Walcraft