XVII Brazilian Symposiun on Programming Languages (SBLP'2013)
In: III Brazilian Conference on Software: Theory and Practice (CBSOFT'2013)

On the Performance of Multidimensional Array Representations in Programming Languages Based on Virtual Execution Machines

Francisco Heron de Carvalho Junior
Cenez Araújo de Rezende
Jefferson de Carvalho Silva
Francisco José Lins Magalhães
Renato Caminha Juaçaba Neto

ParGO Research Group
Pós-Graduação em Ciência da Computação
Universidade Federal do Ceará
Fortaleza/CE, Brazil

MDCC/UFC

Topics

- Context and Motivations;
  - Virtual execution environments (VEE);
  - High performance computing (HPC);
  - Multidimensional array representations;
  - Related Works (VEEs in HPC);
- Goals;
- Experimental Methodolody;
- Results and Discussion;
- Conclusions.

Context and Motivations
High Performance Computing (HPC)

- Computationally intensive algorithms for solving problems in applications of **computational sciences** and **engineering**;
- Use of <u>parallel computing</u> for accelerating computations
  - However, it pressuposes **<u>optimal sequential code</u>**
- Common data structures:
  - Symbolic computations:
    - graphs, in-memory databases, tuple spaces, etc.
  - **Numeric intensive computations:**
    - **multidimensional arrays;**
- Fortran, C and C++ are the preferred languages;
  - Native execution;

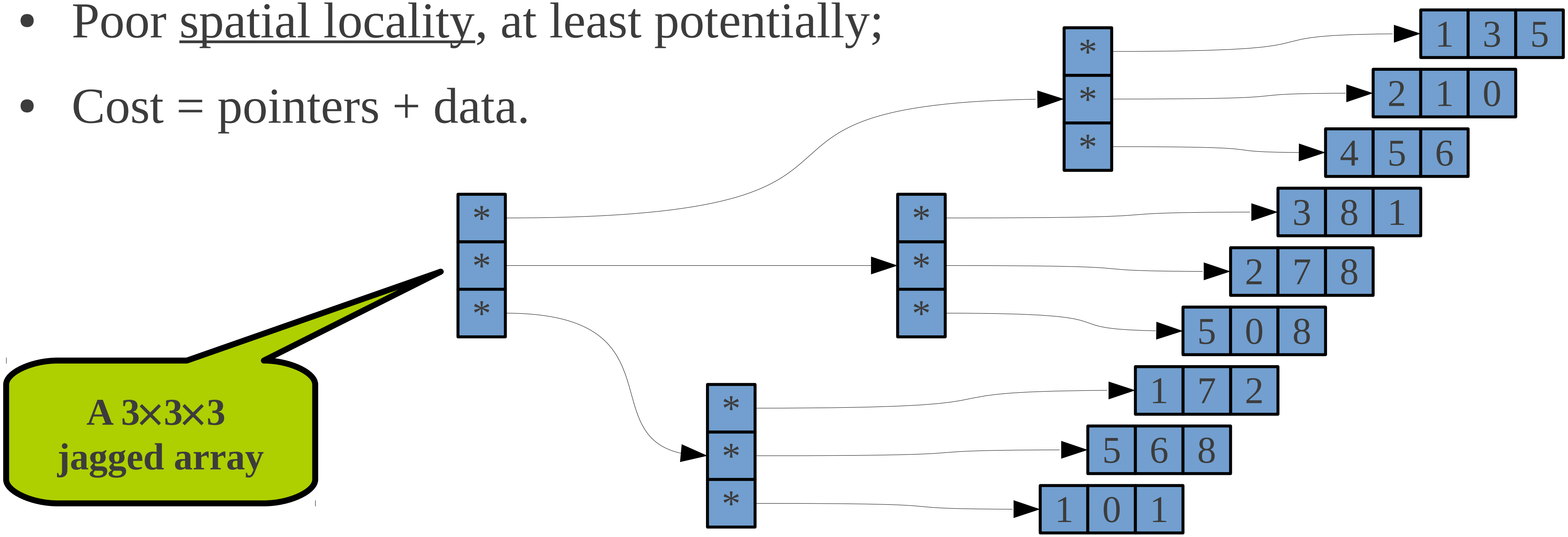
Virtual Execution Enviroments (VEE)

- Abstracting away from hardware and operating systems;
  - Code mobility (web applications, distributed systems);
  - Cross-platform portability;
  - Security;
  - Just-in-time compilation;
- Java Virtual Machine (**JVM**)
  - Mid of 1990s, by Sun Microsystems (now, Oracle Corporation);
  - As intermediate language (IL), the *bytecode*;
- Common Language Interface (**CLI**) – ECMA 334
  - Beginning of 200s, by Microsoft and partners, as an alternative to Java;
  - Support for multiple programming languages; version control; polymorphic IL through CTS (Common Type System); ahead-of-time compilation, rectangular arrays; easy interface with native code; etc

Multidimensional Array Representations in VEEs

Jagged Arrays (Arrays of Arrays)

- The native representation of multidimensional arrays in **Java**;
- An k-dimensions array is an 1-dimensions array whose elements are (k-1)-dimensions arrays;
- Poor spatial locality, at least potentially;
- Cost = pointers + data.



Multidimensional Array Representations in VEEs

Unidimensional Array Embedding

- Trying to improve spatial locality;
- Popular technique among C/C++ programmers;
- Index arithmetic at source code (no compiler-level optmizations);

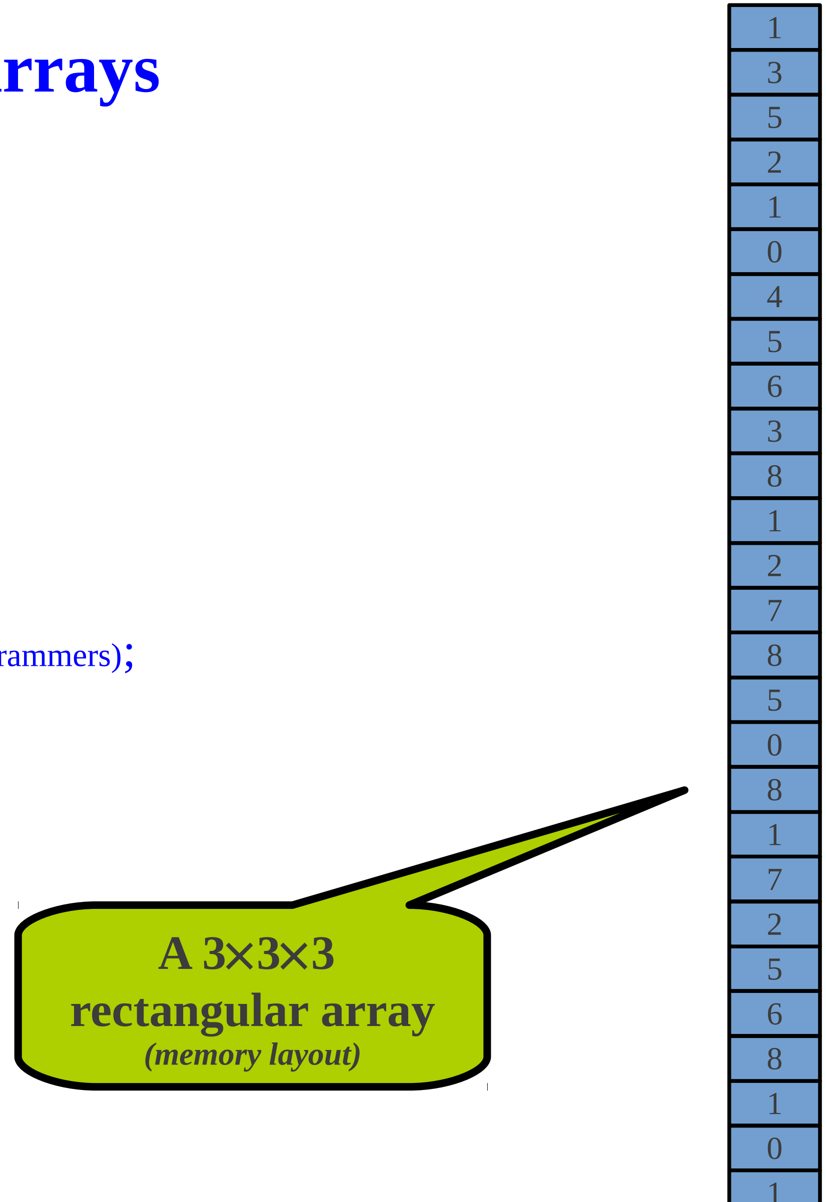
$$index\,(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} \left(x_i * \prod_{j=i+1}^{k} N_j \right)$$

- Why not encapsulating index arithmethic in functions ?
  - HPC programmers tend to avoid function calls overhead (are they correct ?);
  - Code tangling, making difficult code readability and maintainability;

Multidimensional Array Representations in VEEs

Rectangular Arrays

- Fortran !
- Spatial locality;
  - Contiguous memory addresses;
  - Row-major *vs* Column-major traversing;
- Higher level of abstraction;
  - Compiler-level index arithmetic (transparent to programmers);
  - Architecture-specific arithmetic optimizations;
  - Better readability;
- CLI platforms
  - Mono and .NET;
  - C#.
- Cost = data (no pointers).



Context and Motivations

VEEs in HPC (Related Works)

- Java attracted the attention of scientific programmers, with HPC interests, in the end of 1990s;
  - Development productivity and code interoperability;
  - Poor performance of virtual execution, compared to native (C/Fortran);
  - Java Grande Forum – http://www.javagrande.org;
- Several research efforts for making Java efficient for HPC:
  - New abstractions through language extensions ;
  - **Better multidimensional array support;**
  - Highly tuned scientific computing libraries;
  - Interoperability with native languages;
  - Compilers: loop optimizations, array-bound-checking (ABC) elimination.
- **JIT (Just-In-Time compilation)**.

VEEs in HPC (Related Works)

- .NET/CLI attempted to target claims of HPC programmers:
  - Support for rectangular arrays;
  - Programming language agnostic, with interoperability support;
  - Just-in-time compilation;
  - Compiler optimizations;
  - Ahead-Of-Time (AOT) compilation.
- W. Vogels (2003) compared CLI and JVM implementations
  - CLI (Mono and .NET) does not cause significative gains in performance;
  - **Rectangular arrays:** annoying performance compared to jagged arrays;
- Frumkin et al. (2003) proposed **NPB-JAV**
  - Multithreaded **Java** implementation of NAS Parallel Benchmarks;
  - Java still far from Fortran and C;
  - Confirmed by Nikishkov et al. (2003), using OO design in FEM code.

VEEs in HPC (Related Works)

- Riley et al. (2003) and Bull et al. (2001)
  - Java Grande Benchmarks in C and Fortran for comparisons;
  - Slowdown factors between 1.5 and 2.0 in Sun and IBM JVMs.
- Amedro et. al. (2010) and Taboada et al. (2013)
  - NAS Parallel Benchmarks, through NPB-JAV;
  - Small performance gaps between Java and C/Fortran;
  - **One representation of multidimensional arrays**;
  - **Only one virtual machine evaluated: Oracle JVM**;
  - Recommendatton: encapsulate index arithmetic in methods;
    - Contradicting common sense of inlining index arithmetic !
    - **Is this conclusion true for all VMs and array representations ?**
    - The answer is important for HPC programmers take advantage of VMs.
    - **<u>Our preliminar experiments evidenced that the answer is “NO” !</u>**

Goals

- To compare the current performance of JVM and CLI virtual execution environments for different approaches to implement multidimensional arrays;

- To obtain a more realist measure of the performance gap between virtual and native execution;

- To identify bottlenecks in the current implementation of virtual execution environments, regarding their support to multidimensional arrays.

Experimental Methodology
Benchmarking Programs

- NAS Parallel Benchmarks (NPB)
  - Performance evaluation of parallel computers for scientific computing;
  - Sequential and parallel implementations;
  - Kernels (EP, IS, CG, MG, **FT**) and simultated programs (**SP**, **BT**, **LU**, **FT**);
  - Default workloads (problem classes): **S**, **W**, **A**, **B**, **C**, **D**, …
  - Reference implementations in C and Fortran coded by specialists;
  - Java reference implementation: **NPB-JAV**
    - unidimensional array embedding (**AU**);
- Derived versions (Java and C#):
  - Unidimensional array embedding using indexing functions **(AU*)**;
  - Jagged arrays: row-major (**AJR**) and column-major (**AJC**);
  - Rectangular arrays in C#: row-major (**ARR**) and column-major (**ARC**).

Experimental Methodology
Virtual Machines

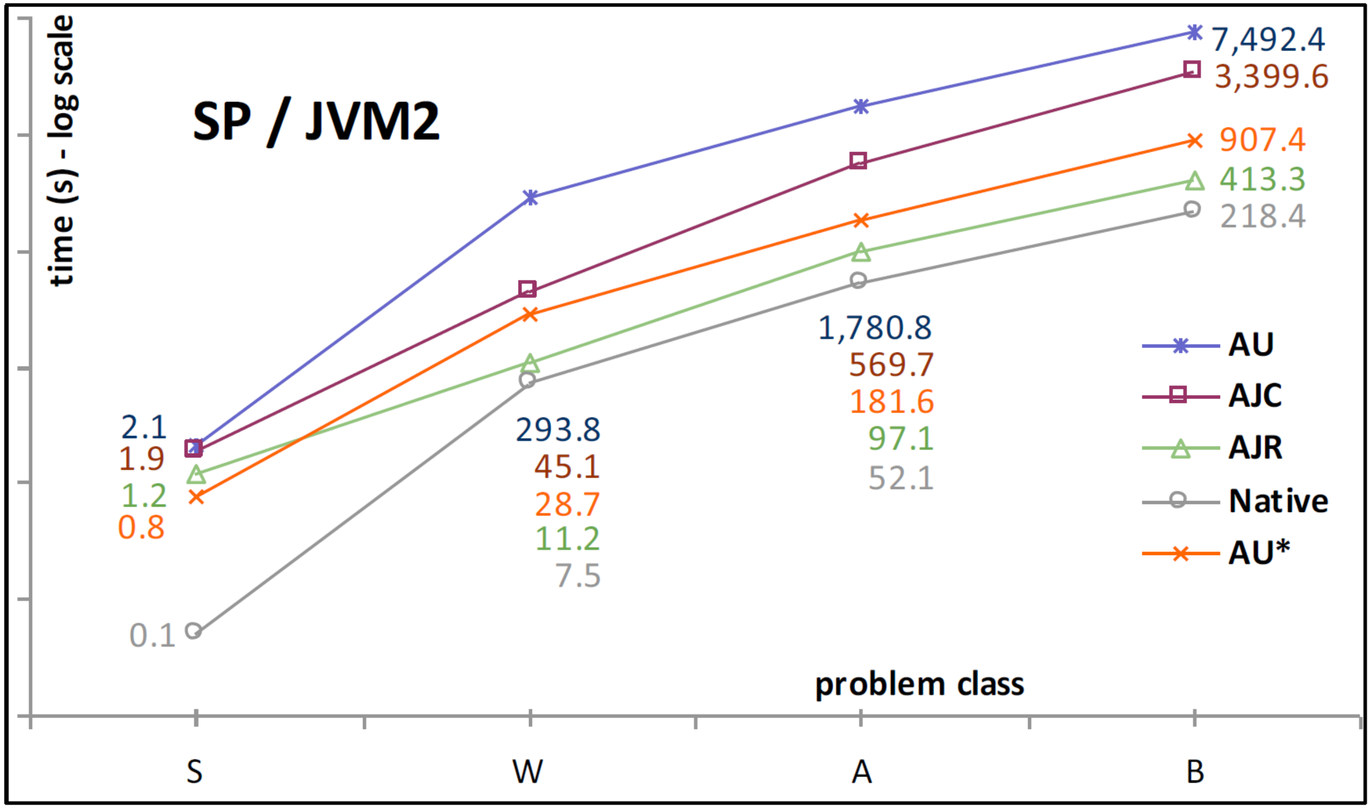
- Java Virtual Machine
  - IBM JVM 1.7 (**$\mathbf{JVM_1}$**)
  - Oracle JVM 1.7 (**$\mathbf{JVM_2}$**)
  - IKVM 0.46.1 (**$\mathbf{JVM_3}$**)
    - JVM implemented on top of Mono/CLI;
- CLI (Common Language Interface)
  - Mono 2.11.3 (**$\mathbf{CLI_1}$**);
  - .NET 4.0.3 (**$\mathbf{CLI_2}$**).

Experimental Cases

- Programs
  - **SP**, **BT**, **LU** and **FT**;
- Problem classes (workloads)
  - **S**, **W**, **A** and **B**;
- Virtual Machines
  - **JVM**$_1$, **JVM**$_2$, **JVM**$_3$, **CLI**$_1$ and **CLI**$_2$;
- Multimensional array representations
  - **AU**, **AU***, **AJC**, **AJR**
  - **ARR**, **ARC**
    - only for **CLI**s.

(P, W, V, R)

Presentation of Results



Also for the other combinations in

$\{$**SP**, **BT**, **LU**, **FT**$\} \times \{\mathbf{JVM_1}, \mathbf{JVM_2}, \mathbf{JVM_3}, \mathbf{CLI_1}, \mathbf{CLI_2}\}$

Using colors and logarithm scale for improving visualization and interpretation

Jagged vs Rectangular Arrays (in CLI)

- **AJR** and **AU** still outperforms **ARR**;
  - After 10 years from the work of W. Vogels, this is still true;

$$\frac{\text{ARR of CLI}_1}{\text{ARR of CLI}_2}$$

$$\frac{\text{ARR of CLI}_2}{\text{Best of All}}$$

	S	W	A	B	S	W	A	B
SP	1.4	1.4	1.4	1.4	3.1	7.0	5.5	5.7
BT	1.7	1.6	1.7	1.7	4.1	9.0	8.8	8.5
LU	1.1	1.5	1.5	1.6	3.9	6.4	6.5	6.1
FT	1.9	1.9	1.9	1.9	3.9	3.9	3.6	3.3
	(a)				(b)			

Results and Discussion

Jagged vs Unidimensional Arrays in JVM

- Contradicting results between Oracle JVM and IBM JVM;
- Oracle JVM
  - Except for FT and for big workloads, AU performs much worse than AJR.
  - AU* greatly improves the performance of AU (index arithmetic in methods);
  - However AU* continues to be worse than AJR by factors varying between 1.8 and 13.4 (avg: ?.?, dev: ?.?)
    - SP / BT / LU × W / A / B;
- IBM JVM
  - AU always performs better than AJR;
  - AU* is worse than AU (cost of method invocations in index arithmetic);
  - **Results of Taboada et al. do not apply to IBM JVM**;
  - Performance figures coherent with current practice of HPC programmers.

IBM JVM *vs* Oracle JVM

- IBM JVM was better when using **AU**, for any workload;
- Oracle JVM was better when using **AJR**, for bigger workloads;
  - Apparently, On Stack Replacement, only affect programs that use jagged arrays;
- **IBM JVM / AU** outperforms **Oracle JVM / JVR** for almost all the cases;
- Difference to native execution between 1.5 and 3.0 for big workloads;
  - Coherence with the results of Riley, et al. (2003)
- For legacy code,
  - use IBM for code based on unidimensional array embedding;
  - use Oracle for code based on jagged arrays.
- For new code,
  - use IBM and AU if you are interested in approaching native performance;
  - use Oracle and AJR if you may to pay on performance to have better code.

Jagged vs Unidimensional Arrays in CLI

- It is not possible to conclude which is better: **AU** or **AJR**;
  - AU is better for FT and BT; AJR is better for SP; indifferent for LU.
- **AU*** causes performance degradation in JVM$_3$, as for JVM$_1$;
- For bigger (**realistic**) workloads, JVM virtual machines (Oracle and IBM) outperforms CLI ones (Mono, IKVM and .NET);

	S	**W**	**A**	**B**	**S**	**W**	**A**	**B**
SP	0.3	2.2	2.4	2.5	0.1	2.4	1.8	2.9
BT	0.4	2.1	2.1	2.1	0.2	3.1	3.7	4.3
LU	0.3	2.1	2.5	2.3	0.2	1.8	1.7	2.9
FT	2.0	1.9	1.9	2.0	1.5	1.4	1.4	–
	(a)				(b)			

Best result of CLIs
Best result of JVMs

- Furthermore, .NET outperforms Mono.

Results and Discussion

Mono Further Optimizations

- Many flags for compiler optimizations (all ON vs all OFF);
  - Only affected **ARR** (10% to 16% of performance improvement);
- Disabling ABC elimination through compiler flags;
  - Only works for jagged arrays (this fact is not documented);
  - For rectangular arrays, we removed ABC code generation in Mini
    - Mono recompiled …
  - ABC overheads are more significatives for jagged arrays than rectangular ones
    - Indiferent to problem class;
    - **AJR:** 29% (**SP**), 26% (**BT**), 33% (**LU**), 33 % (**FT**);
    - **ARR:** 13% (**SP**), 30% (**BT**), 20% (**LU**), 2% (**FT**).

		Optimizations			ABC Off		
		S	W	A	S	W	A
AU	**SP**	1.03	1.03	1.03	1.08	1.07	1.07
	BT	1.01	1.01	1.01	1.15	1.12	1.11
	LU	0.95	1.00	1.00	1.11	1.11	1.11
	FT	0.97	0.97	0.97	1.04	1.04	1.04
AJR	**SP**	1.04	1.04	1.05	1.28	1.30	1.28
	BT	0.98	0.98	0.99	1.26	1.27	1.27
	LU	0.88	1.03	1.03	1.33	1.34	1.32
	FT	0.96	0.96	0.95	1.11	1.12	1.12
ARR	**SP**	1.10	1.10	1.10	1.13	1.14	1.13
	BT	1.16	1.15	1.16	1.31	1.29	1.29
	LU	1.12	1.12	1.13	1.21	1.19	1.20
	FT	1.13	1.14	1.12	1.02	1.02	1.02

Conclusions

- Performance of <u>multimensional array intensive programs</u> vary significantly according to both the underlying <u>virtual machine implementation</u> and the <u>array representation</u>;
- The usual assumption that inline index arithmetic yields better performance due to lower amount of function calls is not valid for Oracle JVM, as pointed out by Taboada et al (2003);

The contrary is valid !

- However, this is not valid for all virtual machines
  - It is not valid for IBM JVM, where inlining improves performance !
- CLI (Mono/.NET) performance is still far from the best JVMs;
- CLI designers must implement rectangular arrays more efficiently !
- Finally, **the results of this paper may help scientific computing programmers on choosing the best virtual machine implementation and array representation, trying to balance their performance and code readability requirements**.

XVII Brazilian Symposiun on Programming Languages (SBLP'2013)
In: III Brazilian Conference on Software: Theory and Practice (CBSOFT'2013)

On the Performance of Multidimensional Array Representations in Programming Languages Based on Virtual Execution Machines

Francisco Heron de Carvalho Junior
Cenez Araújo de Rezende
Jefferson de Carvalho Silva
Francisco José Lins Magalhães
Renato Caminha Juaçaba Neto

ParGO Research Group
Pós-Graduação em Ciência da Computação
Universidade Federal do Ceará
Fortaleza/CE, Brazil

MDCC/UFC