

Re: Fastcode MM B&V 0.41 (extended)

Source:

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.language.basm/2005-05/msg00809.html>

- *From:* "Avatar Zondertau" <avatarzt@xxxxxxxxxx> (please reply to newsgroup)
 - *Date:* 31 May 2005 08:43:39 -0700
-

```
unit SortExtendedArrayBenchmark2Unit;

interface

uses Windows, BenchmarkClassUnit, Classes, Math;

type

TQuickSortExtendedArrayThreads = class(TFastcodeMMBenchmark)
public
procedure RunBenchmark; override;
class function GetBenchmarkName: string; override;
class function GetBenchmarkDescription: string; override;
class function GetSpeedWeight: Double; override;
class function GetCategory: TBenchmarkCategory; override;
end;

implementation

uses SysUtils;

type

TSortExtendedArrayThread = class(TThread)
FBenchmark: TFastcodeMMBenchmark;
procedure Execute; override;
end;

TExtended = record
X : Extended;
Pad1, Pad2, Pad3, Pad4, Pad5, Pad6 : Byte;
end;

TExtendedArray = array[0..1000000] of TExtended;
PExtendedArray = ^TExtendedArray;

function SortCompareExtended(Item1, Item2: Pointer): Integer;
var
```

```
Diff: Extended;
begin
Diff := PExtended(Item1)^ - PExtended(Item2)^;
if Diff < 0 then
Result := -1
else
if Diff > 0 then
Result := 1
else
Result := 0;
end;

procedure TSortExtendedArrayThread.Execute;
var
ExtArray: PExtendedArray;
I, RunNo, Size: Integer;
List: TList;
const
MAXRUNNO: Integer = 8;
MAXELEMENTVALUE: Integer = MAXINT;
MINSIZE: Integer = 100;
MAXSIZE: Integer = 10000;
begin
GetMem(ExtArray, MINSIZE * SizeOf(TExtended));
try
for RunNo := 1 to MAXRUNNO do
begin
Size := Random(MAXSIZE-MINSIZE) + MINSIZE;
ReallocMem(ExtArray, Size * SizeOf(TExtended));
List := TList.Create;
try
List.Count := Size;
for I := 0 to Size-1 do
begin
ExtArray[I].X := Random(MAXELEMENTVALUE);
List[I] := @ExtArray[I].X;
end;
List.Sort(SortCompareExtended);
finally
List.Free;
end;
end;
finally
FreeMem(ExtArray);
end;
FBenchmark.UpdateUsageStatistics;
end;

class function TQuickSortExtendedArrayThreads.GetBenchmarkDescription:
string;
begin
```

Result := 'A benchmark that measures read and write speed to an array of Extendeds. '

+ 'The Extended type is padded to be 16 byte. '
+ 'Bonus is given for 16 byte alignment of array '
+ 'Will also reveal cache set associativity related issues. '
+ 'Access pattern is created by X sorting array of random values using the QuickSort algorithm implemented in TList. '
+ 'Measures memory usage after all blocks have been freed. '
+ 'Benchmark submitted by Avatar Zondertau, based on a benchmark by Dennis Kjaer Christensen. ';
end;

```
class function TQuickSortExtendedArrayThreads.GetBenchmarkName: string;  
begin  
Result := 'QuickSortExtendedArrayBenchmark';  
end;
```

```
class function TQuickSortExtendedArrayThreads.GetCategory:  
TBenchmarkCategory;  
begin  
Result := bmMemoryAccessSpeed;  
end;
```

```
class function TQuickSortExtendedArrayThreads.GetSpeedWeight: Double;  
begin  
Result := 0.75;  
end;
```

```
procedure TQuickSortExtendedArrayThreads.RunBenchmark;  
var  
SortExtendedArrayThread1, SortExtendedArrayThread2 :  
TSortExtendedArrayThread;  
SortExtendedArrayThread3, SortExtendedArrayThread4 :  
TSortExtendedArrayThread;
```

```
begin  
inherited;  
SortExtendedArrayThread1 := TSortExtendedArrayThread.Create(True);  
SortExtendedArrayThread2 := TSortExtendedArrayThread.Create(True);  
SortExtendedArrayThread3 := TSortExtendedArrayThread.Create(True);  
SortExtendedArrayThread4 := TSortExtendedArrayThread.Create(True);  
SortExtendedArrayThread1.FreeOnTerminate := False;  
SortExtendedArrayThread2.FreeOnTerminate := False;  
SortExtendedArrayThread3.FreeOnTerminate := False;  
SortExtendedArrayThread4.FreeOnTerminate := False;  
SortExtendedArrayThread1.Priority := tpLower;  
SortExtendedArrayThread2.Priority := tpNormal;  
SortExtendedArrayThread3.Priority := tpHigher;  
SortExtendedArrayThread4.Priority := tpHighest;  
SortExtendedArrayThread1.FBenchmark := Self;  
SortExtendedArrayThread2.FBenchmark := Self;
```

```
SortExtendedArrayThread3.FBenchmark := Self;  
SortExtendedArrayThread4.FBenchmark := Self;  
SortExtendedArrayThread1.Resume;  
SortExtendedArrayThread2.Resume;  
SortExtendedArrayThread3.Resume;  
SortExtendedArrayThread4.Resume;  
SortExtendedArrayThread1.WaitFor;  
SortExtendedArrayThread2.WaitFor;  
SortExtendedArrayThread3.WaitFor;  
SortExtendedArrayThread4.WaitFor;  
SortExtendedArrayThread1.Free;  
SortExtendedArrayThread2.Free;  
SortExtendedArrayThread3.Free;  
SortExtendedArrayThread4.Free;  
end;
```

end.

• **References:**

- ◆ **Fastcode MM B&V 0.41**
◇ From: Dennis
- ◆ **Re: Fastcode MM B&V 0.41**
◇ From: Avatar Zondertau

- Prev by Date: **Re: filling big array of double**
- Next by Date: **Re: filling big array of double**
- Previous by thread: **Re: Fastcode MM B&V 0.41 (integer benchmark)**
- Next by thread: **Re: Fastcode MM B&V 0.41**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**