



Experience and Lessons Learned for Large-Scale Graph Analysis using GraphX

Jason Dai

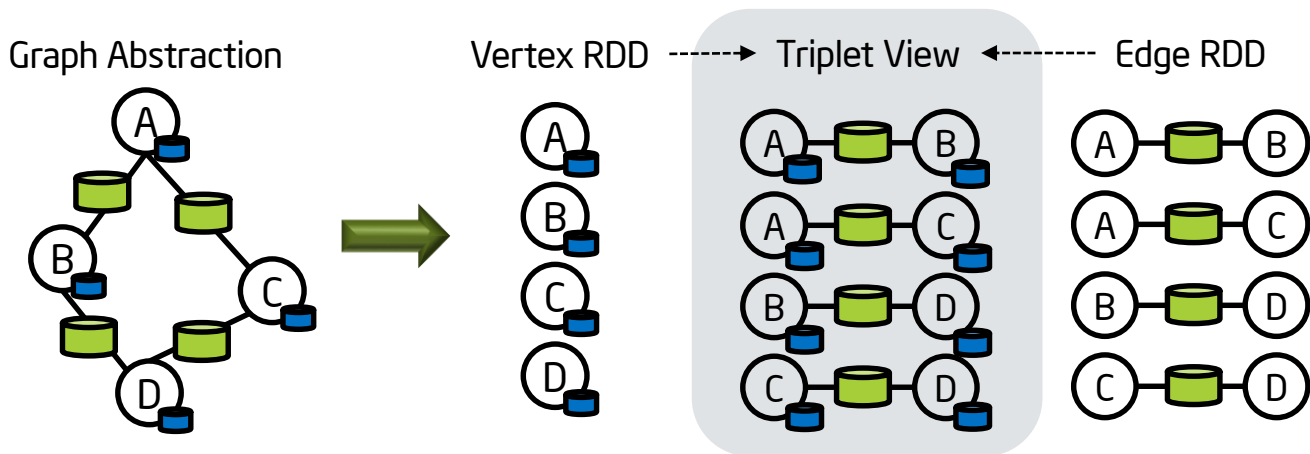
Chief Architect of Big Data Technologies



GraphX Framework

GraphX framework

- Graph parallel computations on Spark data-parallel engine
- Recast graph systems optimizations as distributed dataflow operations
 - Join, view maintenance, etc.



GraphX Applications

GraphX applications

- PageRank

```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```

- Large-scale, iterative Spark applications
 - Billions of edges, 1000s of iterations

Experience applicable to general large-scale iterative Spark applications (read: *machine learning*)

The Dreaded Stack Overflow

Stack overflow error

- First sign of a web-scale problem 😊

```
...
15/03/05 04:14:08 INFO scheduler.DAGScheduler: Job 458 failed: foreachPartition at PageRank.scala:110, took 138.912943 s
Exception in thread "main" org.apache.spark.SparkException: Job aborted due to stage failure: Task 268 in stage 213428.0
failed 4 times, most recent failure: Lost task 268.3 in stage 213428.0 (TID 689532, sr431):
java.lang.StackOverflowError
    at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:1982)
    at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1918)
    ...
15/03/05 04:14:08 INFO scheduler.TaskSetManager: Lost task 32.2 in stage 213428.0 (TID 689524) on executor sr431:
java.lang.StackOverflowError (null) [duplicate 91]
...
```

The Dreaded Stack Overflow

Stack overflow error

- First sign of a web-scale problem 😊

```
...
15/03/05 04:14:08 INFO scheduler.DAGScheduler: Job 458 failed: foreachPartition at PageRank.scala:110, took 138.912943 s
Exception in thread "main" org.apache.spark.SparkException: Job aborted due to stage failure: Task 268 in stage 213428.0
failed 4 times, most recent failure: Lost task 268.3 in stage 213428.0 (TID 689532, sr431):
java.lang.StackOverflowError
    at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:1982)
    at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1918)
    ...
15/03/05 04:14:08 INFO scheduler.TaskSetManager: Lost task 32.2 in stage 213428.0 (TID 689524) on executor sr431:
java.lang.StackOverflowError (null) [duplicate 91]
...
```

- Root cause
 - Serialization of RDD objects with extremely long lineage (due to large iteration# in the program)
- Work-arounds
 - Allocate large JVM stack frame size (i.e., `-Xss`), but suffer from large serialization overheads
 - Checkpoint RDD periodically

Pitfall of RDD Checkpoint

Lazy execution of checkpoint

- Fruitless if marking the RDD for checkpointing after it is materialized

```
//PageRank:  
while (i <- 0 to numIter) {  
  if ((i % 10) == 9)  
    rankGraph.checkpoint()  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
}
```



```
RDD.checkpoint() {  
  ...  
  checkpointData.get.  
    markForCheckpoint()  
}
```



```
SparkContext.runJob() {  
  ...  
  dagScheduler.runJob(...)  
  rdd.doCheckpoint()  
}  
  
RDD.doCheckpoint() {  
  if (!doCheckpointCalled) {  
    doCheckpointCalled = true  
    checkpointData.get.  
      doCheckpoint()  
  }  
  ...  
}
```

Pitfall of RDD Checkpoint

Lazy execution of checkpoint

- Fruitless if marking the RDD for checkpointing after it is materialized

```
//PageRank:  
while (i <- 0 to numIter) {  
  if ((i % 10) == 9)  
    rankGraph.checkpoint()  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
}
```



```
RDD.checkpoint() {  
  ...  
  checkpointData.get.  
    markForCheckpoint()  
}
```



```
SparkContext.runJob() {  
  ...  
  dagScheduler.runJob(...)  
  rdd.doCheckpoint()  
}  
  
RDD.doCheckpoint() {  
  if (!doCheckpointCalled) {  
    doCheckpointCalled = true  
    checkpointData.get.  
      doCheckpoint()  
  }  
  ...  
}
```

Design pattern for managing graph persistence

- `mllib.impl.PeriodicGraphCheckpoint` (call `updateGraph` before the graph is materialized)

```
for (i <- 0 to numIter) {  
  graph = ...  
  graphCheckpoint.updateGraph(graph)  
  ...  
}
```

Pitfall of RDD Checkpoint

“Leakage” of RDD lineage

- Checkpointing breaks long lineage of RDD “dependence”
 - Lineage can still “leak” through reference in RDD member variables/methods

```
class ZippedRDD (var rdd1, var rdd2, ...)
  extends RDD {
  override def compute(part, sc) = {
    ...
    rdd1.iterator(parts(0), sc) zip
    rdd2.iterator(parts(1), sc)
  }

  def clearDependencies() {
    super.clearDependencies()
    rdd1 = null
    rdd2 = null
  }
}
```


Pitfall of RDD Checkpoint

“Leakage” of RDD lineage

- Checkpointing breaks long lineage of RDD “dependence”
 - Lineage can still “leak” through reference in RDD member variables/methods

Design pattern

- RDD reference through dependences whenever possible
- Transient member variable whenever possible
- Clear extra RDD references after checkpointing whenever possible

General fix needed?

- Only RDD.compute required at worker (see SPARK-4672)

```
class ZippedRDD (var rdd1, var rdd2, ...)
extends RDD {
  override def compute(part, sc) = {
    ...
    rdd1.iterator(parts(0), sc) zip
    rdd2.iterator(parts(1), sc)
  }
}
```

```
def clearDependencies() {
  super.clearDependencies()
  rdd1 = null
  rdd2 = null
}
```



```
class ZippedRDD (@transient val rdd1,
  @transient val rdd2, ...) extends RDD {
  override def compute(part, sc) = {
    ...
    dependences(0).rdd.iterator(parts(0), sc) zip
    dependences(1).rdd.iterator(parts(1), sc)
  }
}
```

Costs of RDD Checkpoint

PageRank for Twitter graph

- One iteration: ~100s
- Checkpointing vertex RDD: ~20s
- Checkpointing edge RDD: ~140s

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

A Closer Look at RDD Lineage for GraphX

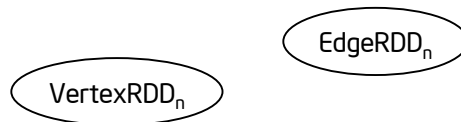
PageRank

```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```

A Closer Look at RDD Lineage for GraphX

PageRank

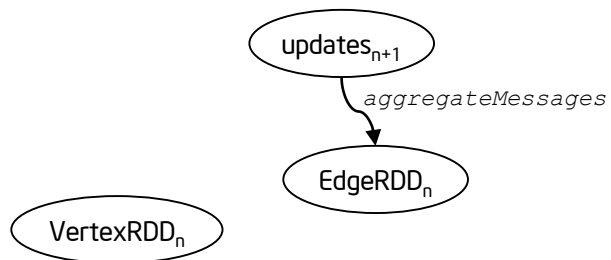
```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```



A Closer Look at RDD Lineage for GraphX

PageRank

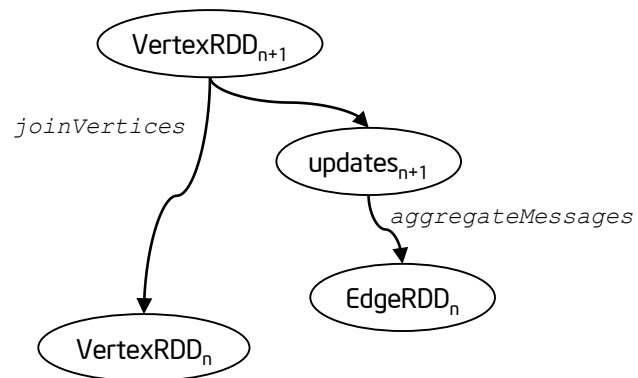
```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```



A Closer Look at RDD Lineage for GraphX

PageRank

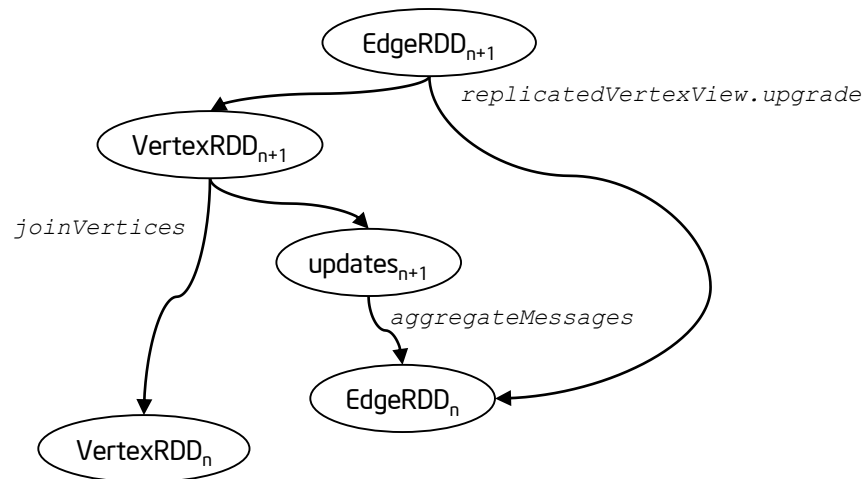
```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```



A Closer Look at RDD Lineage for GraphX

PageRank

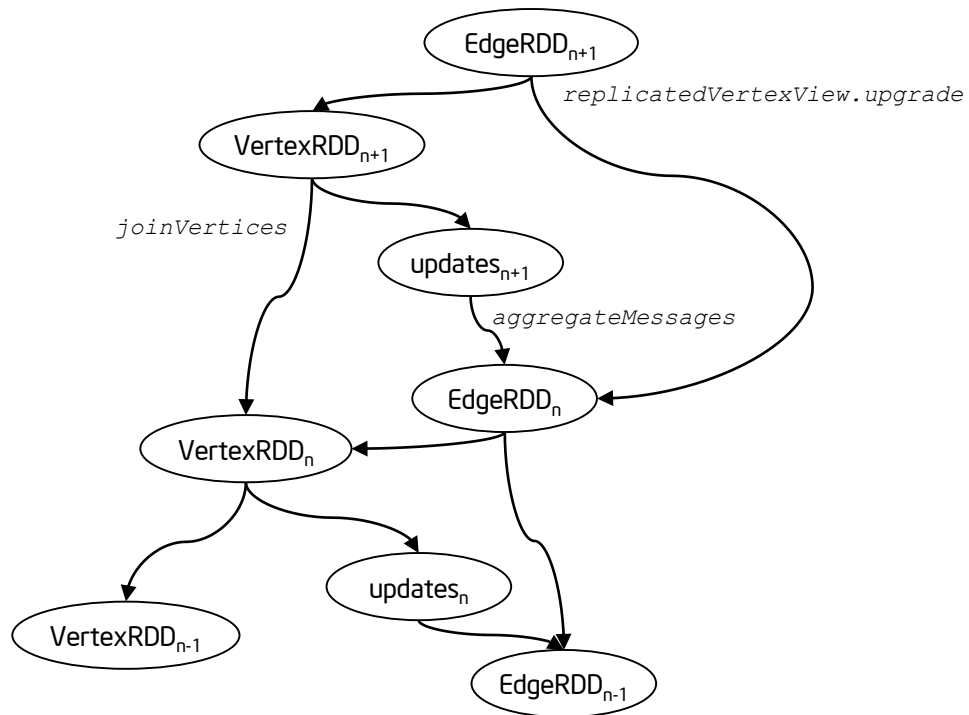
```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```



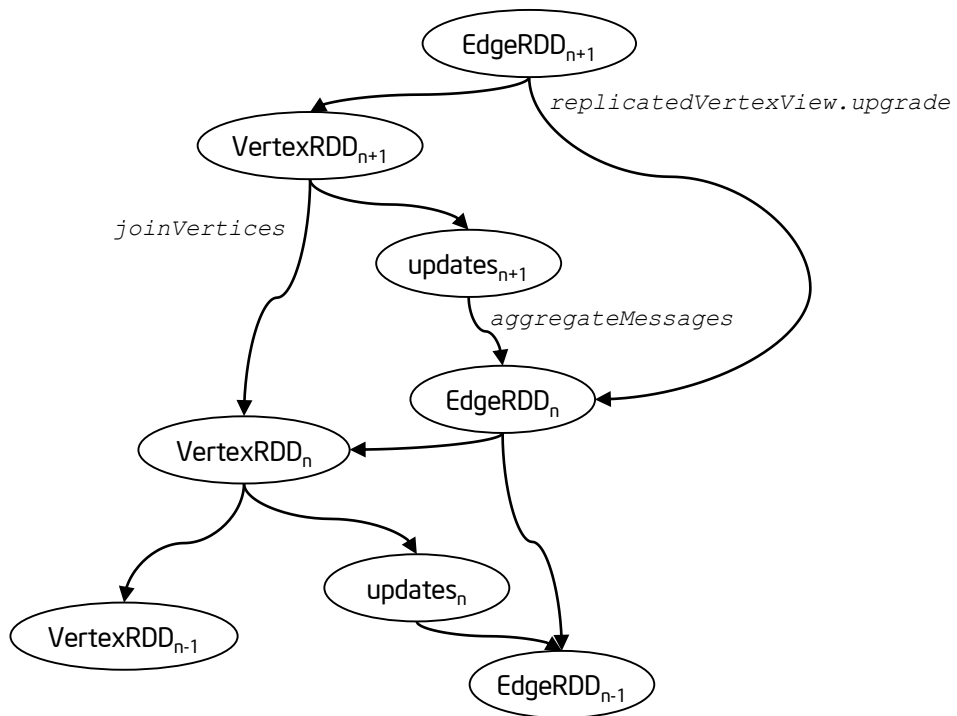
A Closer Look at RDD Lineage for GraphX

PageRank

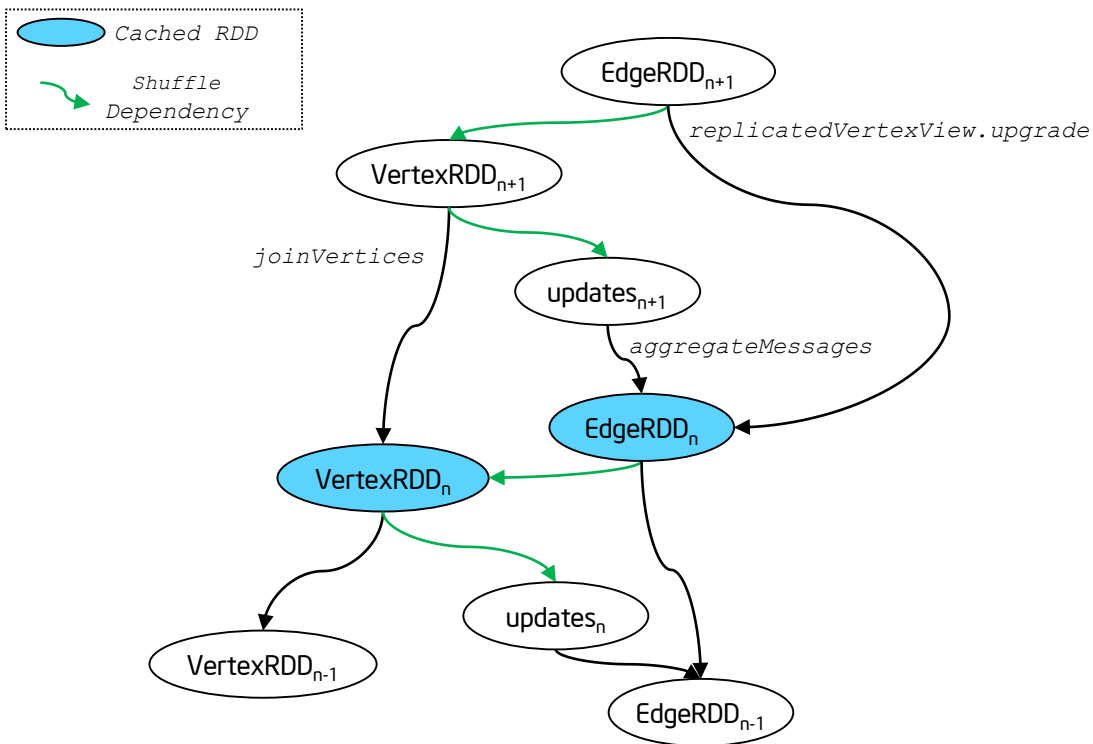
```
while (iteration < numIter) {  
  rankGraph.cache()  
  val updates = rankGraph.aggregateMessages(...)  
  rankGraph = rankGraph.joinVertices(updates, ...)  
  ...  
  iteration += 1  
}
```



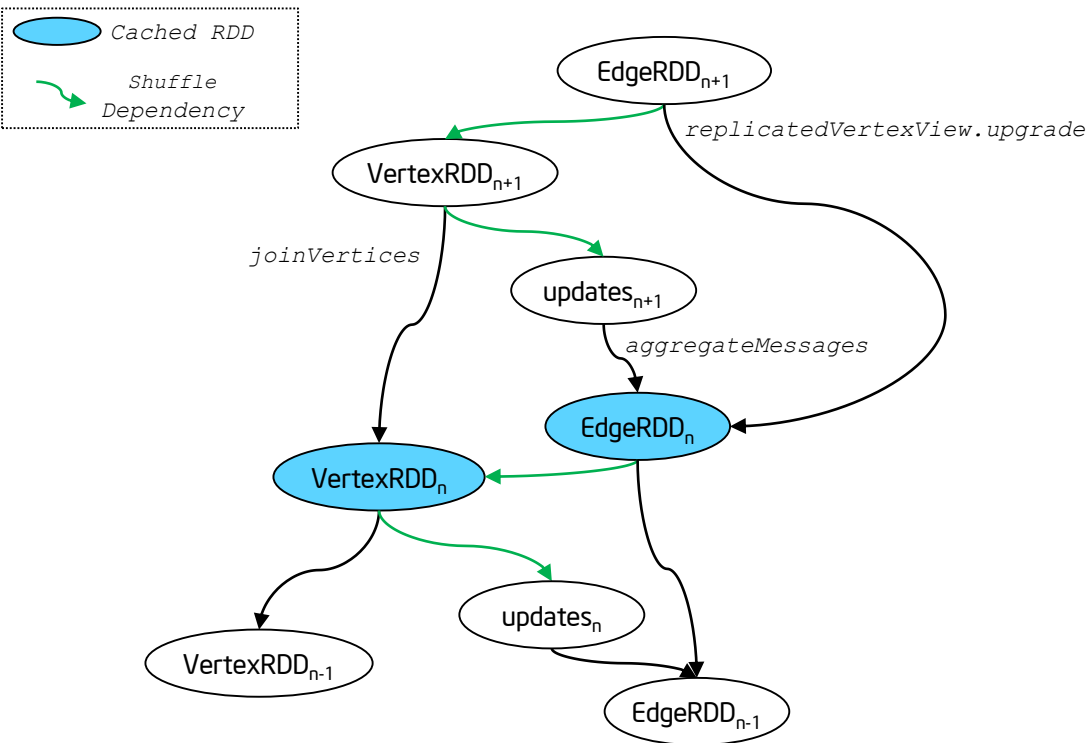
A Closer Look at RDD Lineage for GraphX



A Closer Look at RDD Lineage for GraphX



A Closer Look at RDD Lineage for GraphX



Extremely long RDD lineage

- Vertex and edge chains
 - VertexRDD₀ → VertexRDD₁ → VertexRDD₂ → ...
 - EdgeRDD₀ → EdgeRDD₁ → EdgeRDD₂ → ...
- Result of “graph optimizations”
 - In-place update of vertices and edges
- Possible improvements
 - Leverage the cached RDDs in the chain?
 - Reconstruct replicated vertexes?

Summary

GraphX

- Graph parallel computations on Spark data-parallel engine
 - Recast graph systems optimizations as distributed dataflow operations
- Effective support of web-scale graph applications through careful scaling
 - Billions of edges, 1000s of iterations
 - Applicable to general, large-scale, iterative Spark (e.g., ML) applications

Notices and Disclaimers

- Copyright © 2015 Intel Corporation.
- Intel, the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
See Trademarks on intel.com for full list of Intel trademarks.
- All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.
- Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.
- For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
- Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.
- No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.
- You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- The products described may contain design defects or errors known as errata which may cause the product to deviate from publish.

